MB86290 Series
3D Graphics Library V02
User Manual
The core API

Revision 1.0

1. The specifications in this manual are subject to change without notice. Contact our Sales Department before purchasing the product described in this manual.
2. Information and circuit diagrams in this manual are only examples of device applications, they are not intended to be used in actual equipment. Also, Fujitsu accepts no responsibility for infringement of patents or other rights owned by third parties caused by use of the information and circuit diagrams.
3. The contents of this manual must not be reprinted or duplicated without permission of Fujitsu.
4. If the products described in this manual fall within the goods or technologies regulated by the Foreign Exchange and Foreign Trade Law, permission must be obtained before exporting the goods or technologies.

Updated history		
Date	Version	Updated content
2003/1/8	1.0	First edition

（Page of updated history：1／1）

Preface

¦ Purpose of this manual

This manual explains the function of *MB86290 Series 3D Graphics Library V02 –The core API* (3DGL core API) and the application interface to an engineer engaged in developing the application of MB86290 Series Graphics Controller (Graphics Controller).

This manual explains on the assumption that the readers have the knowledge of computer graphics. The details on this are explained on the following item, *the knowledge on graphics to be required to the reader.*

Also, the knowledge required for the development of the application is explained on the following item, *preparations for developing the application.*

¦ The knowledge required to the reader on

It is described the representative techniques handled on this manual and the knowledge required to the reader. To acquire those techniques, please refer to books and documents on public.

Still, this manual uses dher technical term, those are acquired in the process of comprehending the following contents.

【Graphics techniques on this manual】

- Coordinates and coordinate conversion

  3DGL core API converts the geometrical figure defined as the 3D coordinate (the objective coordinate) to the 2D coordinate through the model view conversion (modeling conversion and view conversion), the projection conversion, view port conversion, and draw it.

  Those conversions should be done in drawing the 3D coordinate figure. To use 3DGL core API, the user should understand those conversion processes and is required the knowledge to apply it.

- Lighting

  The lighting is the function of expressing the object with lighting up by the illumination.

  In case of using the function of lighting, it is required the knowledge of the normal vector and calculating the value.

- Shading

  The shading is the technique of erasing the back of 3D object and hidden face behind another object in advance.

  3DGL core API uses the curling and the depth test for the shading. These knowledge and the related terms, such as the depth buffer, are required to understand.

- Texture mapping

The texture mapping is the technique of mapping a bit-mapped image on the surface of the object.

Also, the technique of mapping a picture as converting the object is called the wrapping, and the method is called the wrapping method.

In case of using texture mapping, it is required to suitably specify the coordinate of the texture and the method of wrapping.

¦ Preparations for developing the application

In order to draw a figure using the graphics controller in applications, it is required to initialize the graphics controller, set up the display, and control the display list. These processes are not performed in 3DGL core API, therefore *MB86290 Series Graphics Driver* (Graphics Driver) should be used.

In executing these processes, refer to the following *the material on the specification of the graphics controller* and *the material on the graphics driver.*

¦ The material on the specification of the graphics controller

Refer to Table1 on the specification of graphics controller depended on user's graphics controller.

Table1 Material list

Graphics controller	Name of the material
MB86291/86291S	MB86291 <SCARLET>Specification of graphics controller
MB86291A	MB86291A <SCARLET2> Specification of graphics controller
MB86292/86292S	MB86292 <ORCHID> Specification of graphics controller
MB86293	MB86293 <CORAL-LQ> Specification of graphics controller
MB86294	MB86294 <CORAL-LB> Specification of graphics controller

Notice) There is the case to name specific graphics controllers as the following in this manual.

MB86291, etc	: MB86291, MB86291S, MB86291A, MB86292, MB86292S, MB86293, MB86294
MB86291/86292	: MB86291, MB86291S, MB86291A, MB86292, MB86292S
MB86293, etc	: MB86293, MB86294

¦ The material on the graphics driver

Refer to the following list on the specification of the graphics driver and the programming.

MB86290 Series Graphics Driver V02 User Manual

¦ The notation on the manual

The notation on the manual keeps the following rules, excluding the programming code.

Bold	Function name and constant label (macro in C language and constant)
Italic	Parameter name (parameter)
——	Control flow, such as function call (showed in diagram)
➡	Data flow (showed in diagram)
□	Software or process (showed in diagram)
▢	Data (showed in diagram)

Also, notation ended by an asterisk, such as **glColor3***, shows the function family, that are same function, but are different data type of the parameter. For example, in case of **glColor3***, it shows the following function family.

glColor3f、**glColor3fv**、**glColor3i**、**glColor3iv**、**glColor3ui**、**glColor3uiv**、
glColor3ub、**glColor3ubv**

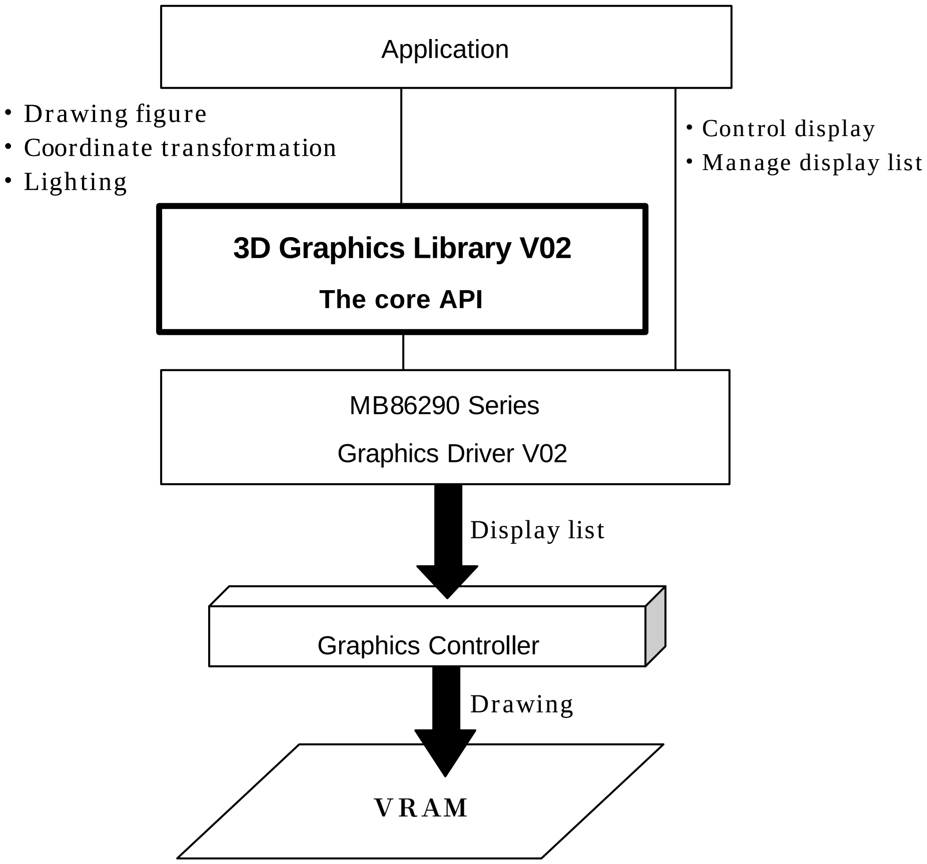
Content

1. GENERAL

MB86290 Series 3D Graphics Library V02 –The core API (3DGL core API) is a function family to support the development of 3D graphics application (application) utilizing MB86290 Series Graphics Controller (graphics controller), and provides the fundamental function of graphics, such as the coordinate conversion, lighting arithmetic, drawing of primitive object (line, triangle, etc).

1.1. Program structure

Figure1.1 shows the program structure utilizing 3DGL core API.



1.1 Program structure

3DGL core API stands between applications and *MB86290 Series Graphics Driver V02* (Graphics Driver), and provides the higher level interface for 3D graphics drawing than the graphics driver for applications.

3DGL core API makes applications release from the complexes process, such as the matrix calculation for the coordinate transformation, and to use added function at the same time, such as the lighting.

The other hand, 3DGL core API does not provide any function except 3D graphics drawing. It is not required to use the graphics driver for the various settings of the graphics controller and the management of the display list, commencing with the screen display.

1.2. The flow of process

Figure1.2 shows the flow of 3DGL core API process.

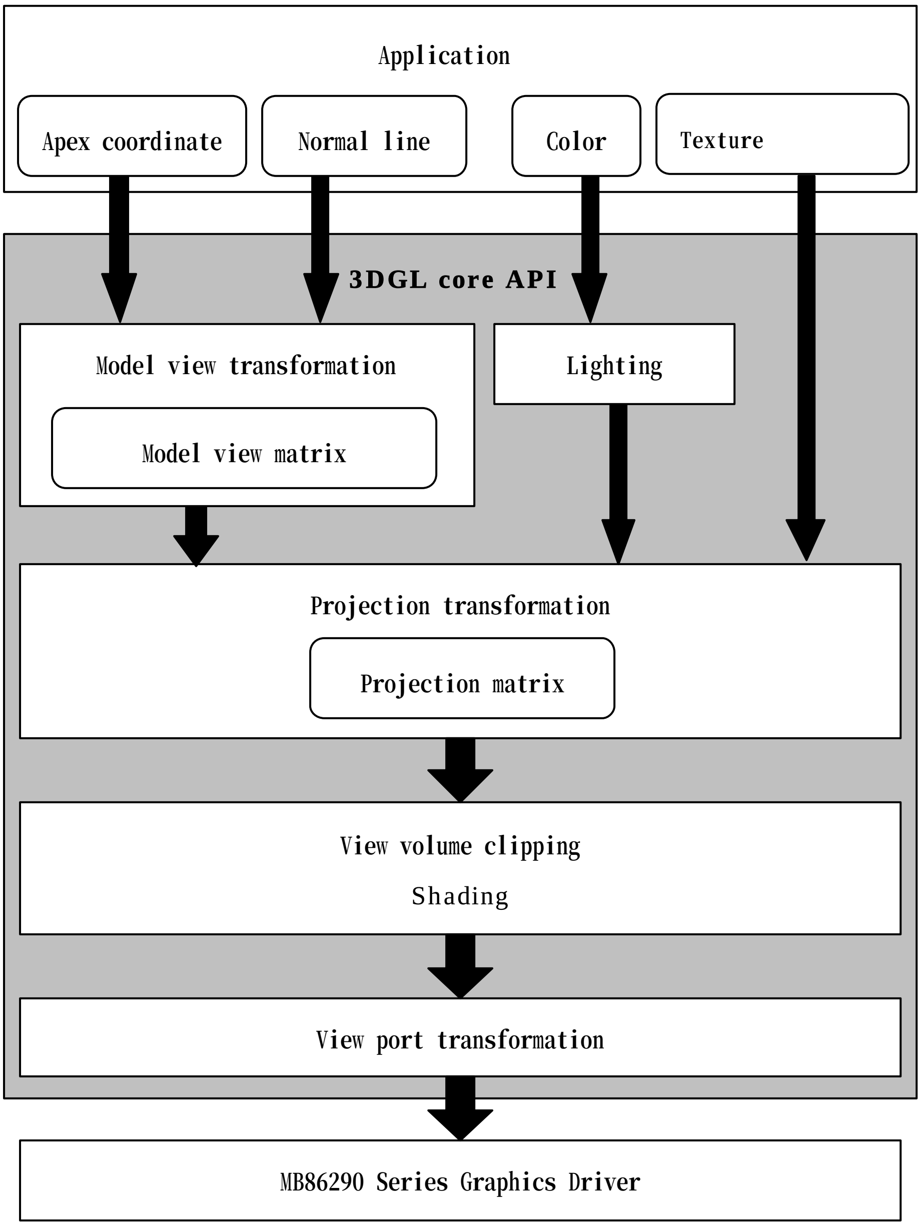


Figure1.2 The flow of process

1.3. Operating conditions

Each function of 3DGL core API operates in following conditions

- The memory of graphics (graphics memory) is mapped I the CPU address field.
- The driver functions are practicable to be executed. (The driver context, DL buffer is formed.)

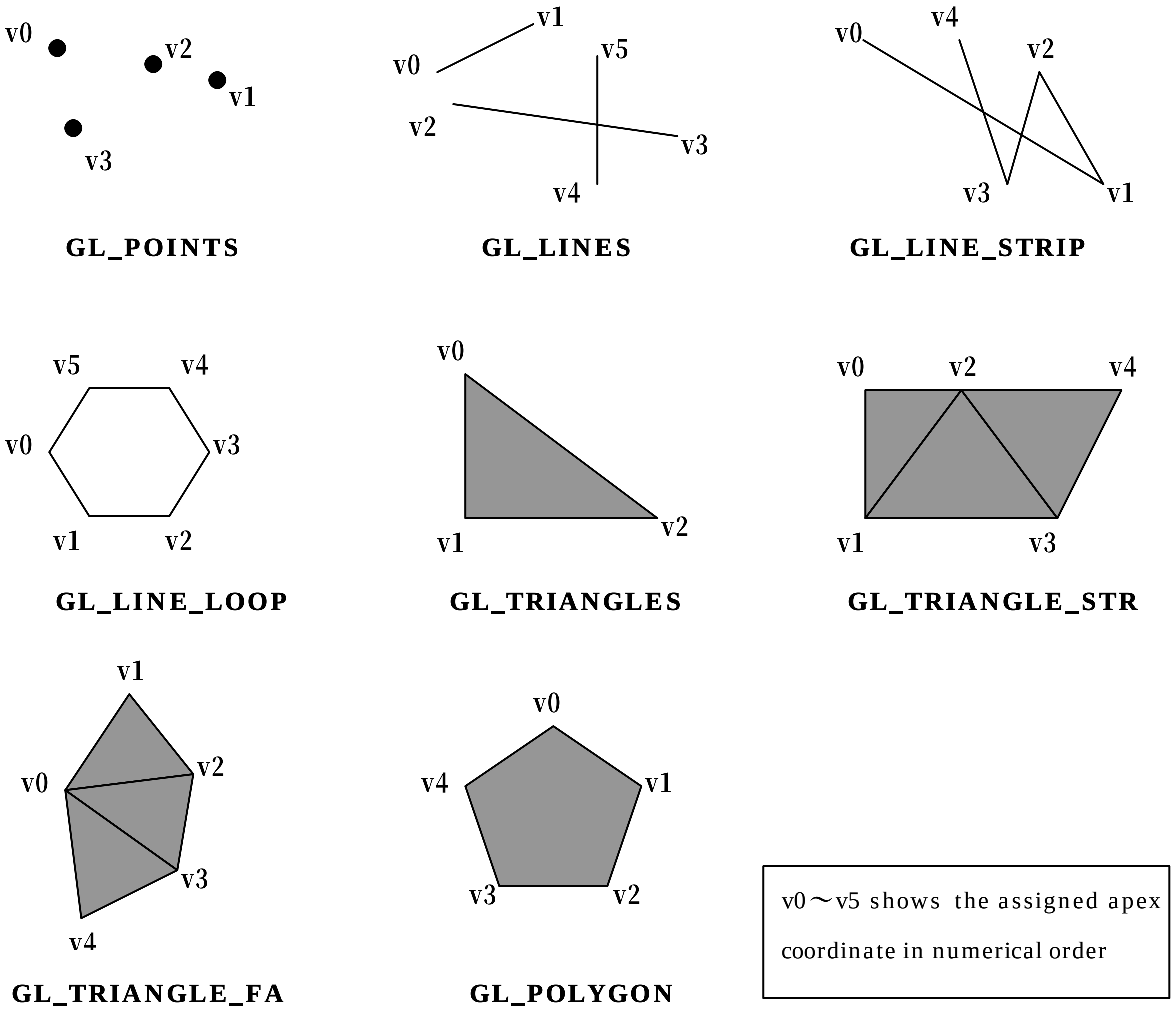
2. The function of 3D Graphics Library core API

This chapter explains the functions of 3DGL core API. The function to use each function is described as *the function to use this function*. For a method to use each function, refer to *3D Graphics Library core API reference.*

2.1. Primitives

The primitives are a basic graphic form, which makes up the object.

It shows usable primitives by 3DGL core API in Figure2.1. The description of each primitive is described in Figure2.1.



v0～v5 shows the assigned apex coordinate in numerical order

Figure2.1 Primitives

It explains each primitive in the following.

■ **GL_POINTS**

It draws a point on each apex coordinate.

- **GL_LINES**

It draws a line segment, which connects two apexes in the assigned order.

If a total of apex is an odd number, last apex is ignored.

- **GL_LINE_STRIP**

It draws a line segment, which connects all apexes coordinate in the assigned order.

A line can be intersected freely.

It does not draw anything, when a total of apex is less than one.

- **GL_LINE_LOOP**

It draws a loops line, which connects first apex and last apex of **GL_LINE_STRIP**.

- **GL_TRIANGLES**

It draws a triangle, which connects three apexes in the assigned order.

The assigned order of the apex determines the front face/ the reverse face. In the default, when the assigned order of the apex is counter clockwise, it becomes the front face. (The triangle shown in Figure2.1 is the front face, as v0? v1? v2 is counter clockwise.)

If a total of apex is not in multiples of three, extra apexes are ignored.

- **GL_TRIANGLE_STRIP**

It draws a triangle, which is arranged in a transverse direction.

The drawing order of **GL_TRIANGLE_STRIP** in Figure2.1 is that, first it draws a triangle, which connects v0, v1, and v2, and following it draws a triangle, which connects v2, v1, and v3, and then it draws a triangle, which connects v2, v3, and v4. From then on, every time it adds an apex, it adds a triangle, which connects three apexes specified at last. Still, as well **GL_TRIANGLES**, the assigned order determines the front face/ the reverse face. The order, which connects an apex of each triangle, becomes all same direction (either clockwise or counter clockwise).

- **GL_TRIANGLE_FAN**

It draws a triangle, which is arranged in alary.

The drawing order of **GL_TRIANGLE_FAN** in Figure2.1 is that, first it draws a triangle, which connects v0, v1, and v2, and then it draw v0, v3, v4. From then on, every time it adds an apex, it adds a triangle, which connects two apexes, first specified apex, v0 and last specified apex. Still, as well **GL_TRIANGLES**, the order of the apex determines the front face/ the reverse face. The order, which connects apexes of each triangle, becomes all same direction (either clockwise or counter clockwise).

- **GL_POLYGON**

It draws a polygon, which connects all apex coordinates in the assigned order.

It is necessary that a polygon is convexity. If the apex data is entered to be convexity, the result is not guaranteed.

At least three apexes are required for a total.

【Functions to use this features】

The primitive is drawn by a combination of **glBegin** function and **glEnd** function. Each apex coordinate of the primitive is specified by calling **glVertex3*** function by between **glBegin** function call and **glEnd** function call. **glVertex3*** function specifies xyz-coordinate of one apex. If it draws a triangle, it calls **glVertex3*** function three times and specifies three apexes.

In **glBegin** function and **glEnd** function, it is possible to call **glColor3*** function (color input), **glNormal3*** function (configuration of normal line vector), **glTexCoord2*** function (texture coordinate input), besides **glVertex3***. If another function is called, it becomes error.

2.2. Color

There are the following primary colors treated in 3DGL core API.

- Apex color
- Illumination color on the lighting（refer to *2.7 Lighting*）
- Pixel color of the texture image（refer to *2.8 Texture mapping*）
- Border color（refer to *2.8.4 Texture wrapping*）

The apex color is color of each coordinate of the primitive, and it is used for the shading (refer to *2.3 Shading*).

The illumination color is color of the light, which illuminates the object, and it is used for the shading.

The pixel color of the texture image is color of each pixel, which makes up a texture image. The texture image is a bit-mapped image data, which is used in the texture mapping.

There are RGB mode and RGBA mode for a color format. It explains in detail in the following.

■ RGB mode

The RGB mode is color, which is made up by red, green, and blue elements.

Each element has the value of [0.0, 1.0] range. In API function, which specifies RGB as the integer type, it performs a linear mapping, which corresponds to the decimal type internally, the minimum of the integer type is 0.0, and the maximum is 1.0.

■ RGBA mode

In the RGBA mode, the alpha value is added besides red, green, and blue.

In specifying the border color, the alpha value is used to determine the alpha bit value (refer to *2.8 Texture mapping*). The alpha value in specifying the illumination and the material color is provided for the extension in future. There is a function, which specifies the alpha value on the API specification, however it does not use the alpha value in the internal process.

【Functions to use this features】

The function, which is used to specify the each color, is the following.

■ Apex color

glColor3* function is used to specify the apex color. Color specified by **glColor3*** function is set as *the current color*, and the current color is used until it is set again.

■ Illuminating color

glLight* function is used to specify the illumination color.

- **Material color**

  glMaterial* function is used to specify the material color.

- **Border color**

  glTexParameter* function is used to specify the border color.

2.3. Shading

The shading is the process, which colors the face of a triangle or a polygon. The shading method of 3DGL core API is the flat shading and the smooth shading (glow shading). It explains each shading method in the following.

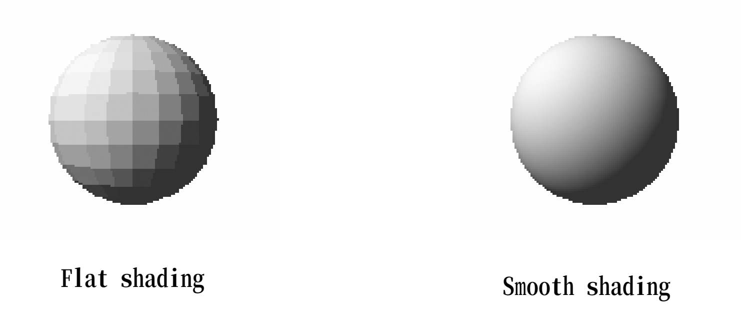
- **Flat shading**

  It performs the shading in mono-color. As each drawing primitive, which makes up the object, is mono-color, the joint line of the drawing primitive is visibility on the object surface.

- **Smooth shading**

  It specifies different color on each apex, and it performs the shading with the linear interpolating of color between each apex. As color of each drawing primitive is varied smoothly, it looks the surface of the object round.

Figure2.3 shows an example of the shading performed by each shading method. The object is 3D body, which is made up by a combination of the drawing primitive, and in Figure2.3 the spherical body is made up by a combination of a polygon.



Flat shading

Smooth shading

Figure2.3 Example of shading

【Functions to use this features】

glShadeModel function is used to specify the shading mode. It does not operate to draw a point and a line. These are always drawn in mono-color.

2.4. Alpha blending

The alpha blending is function, which expresses permeability by blending two colors.

If the alpha blending is used, the drawing primitive is blended with the original color on the drawing frame and one pixel unit. As the result, previously drawn image is transparent under the figure, which is drawn by the drawing primitive.

From then on, it calls the alpha blending just *blend*.

【Functions to use this features】

The alpha blending selects valid or invalid by **glEnable** function and **glDisable** function.

Also, it specifies a mixture ratio of color (blend index) by **glAlpha*** function. The value specified by **glAlpha*** function is configured in *the current blend index* and it keeps the value until next configuration. It is necessary to specify the blend index before performing **glBegin** function.

2.5. Model view transformation

The model view transformation is an operation performed through the modeling transformation and the view transformation.

It explains the modeling transformation and the view transformation in the following.

■ Modeling transformation

The modeling transformation is an operation, which arranges the object defined in the local coordinate system (object coordinate system) to the world coordinate system.

The modeling transformation has a rotation, a movement, and a zooming. It configures the position and size of the object by a combination of these operations.

■ View transformation

The view transformation is an operation, which configures the visual axis direction toward the object.

The view transformation has a rotation and a movement. It configures the position of the visual axis and the angle of the visual axis. However, the eye view and the visual axis are conceptualistic; practically the object movement enables to configure the eye view and the visual axis. Therefore, it performs the modeling transformation and the view transformation all together.

【Functions to use this features】

A rotation, a movement, and a zooming for the model view transformation are performed using **glRotatef** function, **glTranslatef** function, and **glScalef** function perceptively.

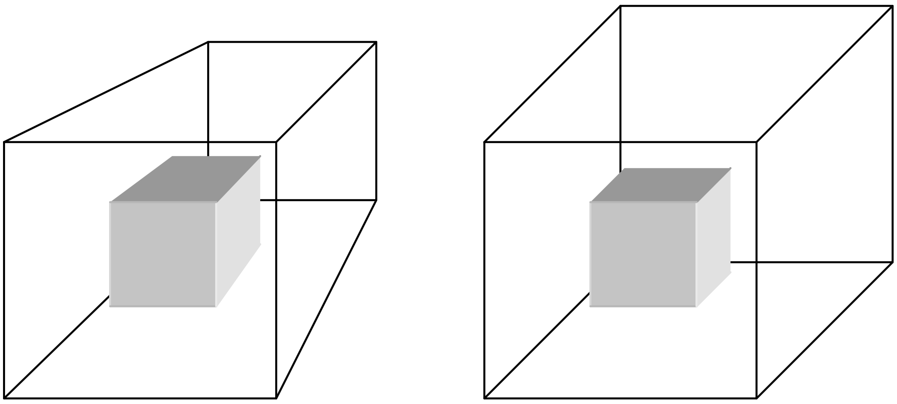
2.6. Projection transformation

The projection transformation is an operation, which determines that how the object defined in 3D is projected on the screen in 2D.

There are the perspective projection and the orthogonal projection for the projection. Figure2.6a shows an example of drawing in each projection.

The perspective projection is used to express a perspective as same as the real vision. It shows the object smaller as the object is further.

The orthogonal projection shows the object independently of the distance from the eye view. Therefore, it can perform the accurate reflection of size and shapes of the object.



Perspective projection

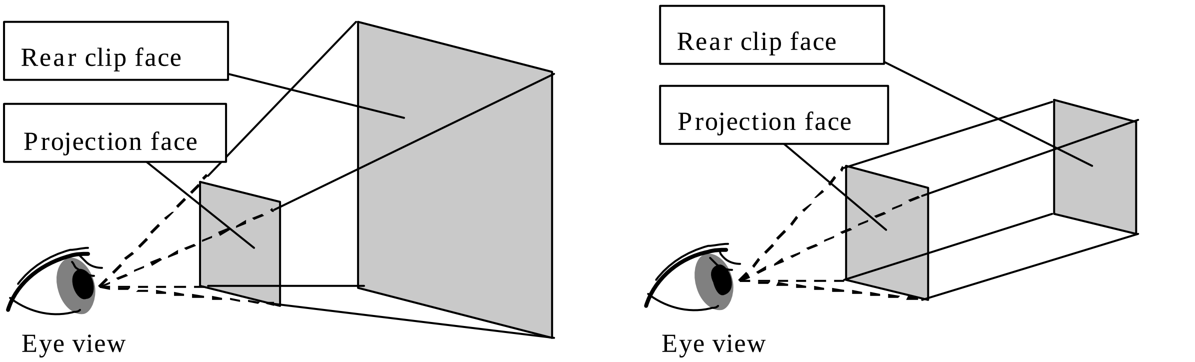
Orthogonal projection

Figure2.6a Perspective projection and orthogonal projection

It specifies the space (view volume), which can see from the eye view, in specifying the projection transformation. The view volume in specifying the perspective projection is shown in Figure2.6b(A), and the view volume is a rectangular frustum, which is made up by the projection face and the rear clip face. Also, the view volume in specifying the orthogonal projection is shown in Figure2.6b(B), and the view volume is a rectangular solid, which is made up by the projection face and the rear clip face.

Both of the perspective projection and the orthogonal projection use the parameter shown in Figure2.6c for specifying the view volume.

“near” is the distance between the eye view and the projection face (length of perpendicular, which is dropped from the eye view to the projection face). “far” is the distance between the eye view and rear clip face (length of perpendicular, which is dropped from the eye view to the rear clip face). (left, top) and (right, bottom) show the upper left coordinate and the lower right coordinate on the projection face, when the intersection of the eye view direction and the projection face is the origin.



(A) View volume in the perspective projection (B) View volume in the orthogonal projection

Figure2.6b View volume

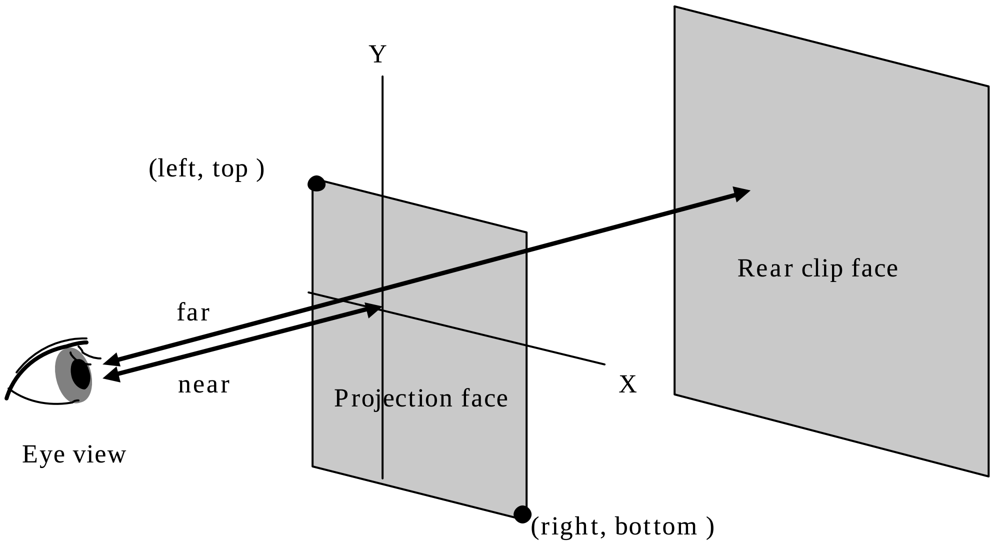


Figure2.6c Parameter of the view volume

【Functions to use this features】

It specifies the perspective projection and orthogonal projection by **glFrustum** function and **glOrtho** function respectively.

2.7. Lighting

The lighting (illuminating process) is a function, which illuminates the object to bright out the scene (full view) realistically. It can use maximum eight illuminations at the same time in the lighting.

It explains the lighting in detail in the following.

2.7.1. Usage of the lighting

To perform the lighting, it configures *illumination parameter*, *material parameter*, and *illumination model parameter*.

The illumination parameter shows the characteristic of the light, and it is specified for each illumination. The material parameter shows the reflection characteristic of the light, and it is specified as the replacement for the apex color for each illumination. The illumination model parameter shows the process method in calculating the object color.

Figure2.7.1 shows an example of the lighting.

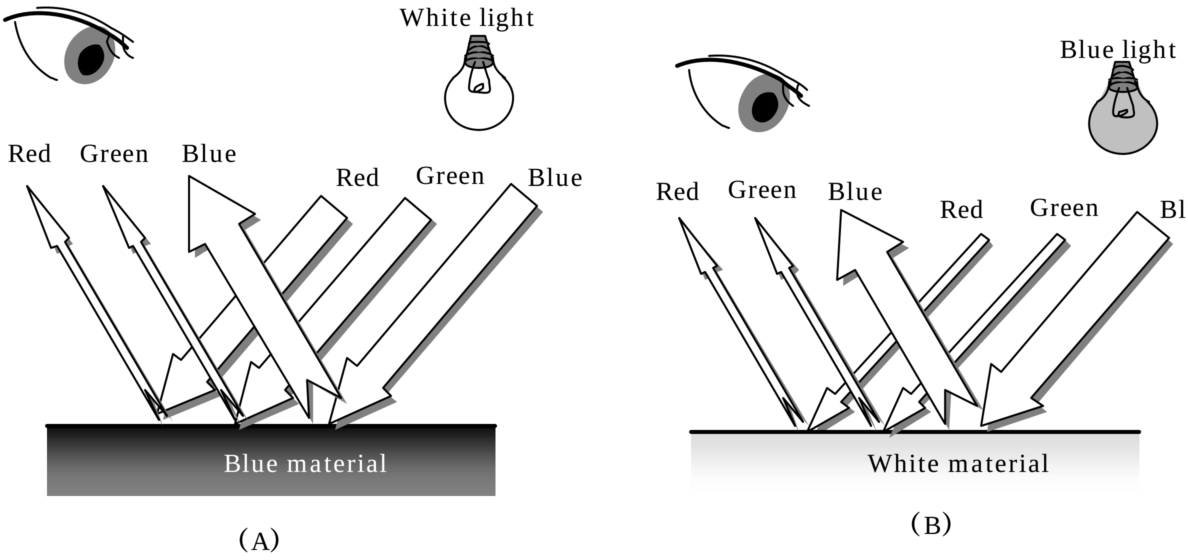


Figure2.7.1 Color of object in lighting

Figure2.7.1(A) is an example of configuring a white light and a material, which reflects a blue light. In this example, as only blue light is reflected out of all lights, the object looks blue. The other hand, Figure2.7.1(B) is an example of configuring a blue light and a material, which reflects all lights. In this case, as the illuminating color is blue, the object looks blue, not white. As just described, the object color in lighting is determined by calculating the illumination parameter and the material parameter based on the illuminating model parameter.

The illumination parameter, the material parameter, and the illumination model parameter have the elements shown in Table2.7.1. The details are explained in the following.

Table2.7.1 Element of each parameter on the lighting

Parameter	Component
Illumination parameter	Cutoff
	Projection direction
	Diffused light
	Secular light
	Ambient light
	Attenuation constant
	Brightness distribution index
Material parameter	Reflection coefficient of diffused light
	Reflection coefficient of specula light
	Reflection coefficient of ambient light
	Emitted light
	Specula brightness distribution index
Illumination model parameter	Reflection direction of specula light
	Ambient light of full view
	Material, which is used for the lighting calculation of rear face

【Functions to use this features】

Valid/Invalid of the lighting is specified by **glEnable** function and **glDisable** function respectively.

The configuration of the illumination parameter is performed by **glLight*** function.

The configuration of the material parameter is performed by **glMaterial*** function.

The configuration of the illumination model parameter is performed **glLightModel*** function.

2.7.2. Cutoff and projection direction

The cutoff shows limits on the projection angle for the light projection direction as shown in Figure2.7.2a. The value of cutoff is 0~90°, or 180°. If it specifies 0~90°, the light becomes the spot light as shown in Figure2.7.2b(A). The spot light illuminates the face, which is the inside of a circular cone specified by the cutoff. If it specifies 180° for the cutoff, the light becomes the point light as shown in Figure2.7.2b(B). The point light illuminates in all directions uniformly.

The projection direction is the parameter, which shows the light direction form the light. It can specify the parameter in only the spot light.

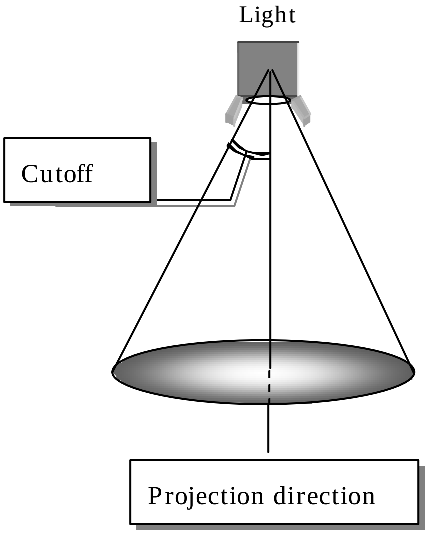


Figure2.7.2a Cutoff and projection direction

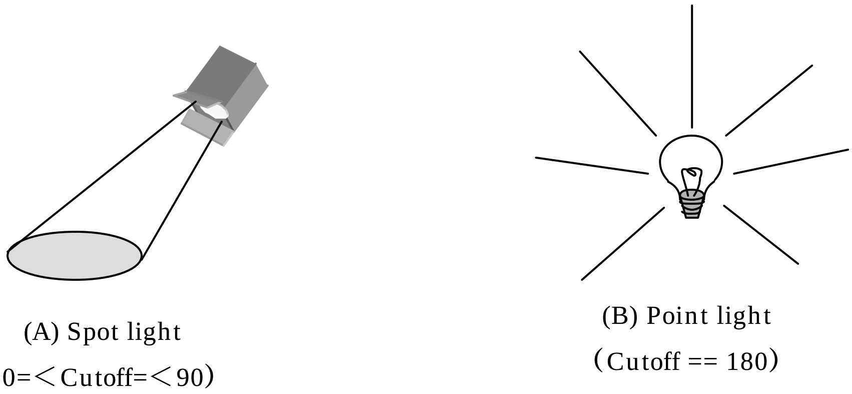


Figure2.7.2b Cutoff and light various

【Function to use this features】

The configuration of the cutoff and the projection direction is performed by **glLight*** function.

2.7.3. Diffused light, specula light, ambient light

The diffused light, specula light, and ambient light show the characteristic of the light. It can specify the light color and the brightness of Red, Green, and Blue for these illumination parameters. It shows the characteristic of each light in the following.

- **Diffused light**

  It is the light of the diffused reflection on the object surface. The brightness on the object surface is varied depended on the incident angle of the light for the object face. (Refer to Figure2.7.3(A))

- **Specula light**

  It is the light of the directional reflection on the object surface. The brightness on the object surface is varied depended on the relation of the reflection light and the eye view. It becomes a component, which determines color of object shining face. (Refer to Figure2.7.3(B))

- **Ambient light**

  It is the light, which is diffused depended on the environment. It illuminates all faces of the object with the same brightness. It becomes a component, which determines color of the shading face. (Refer to Figure2.7.3(C))

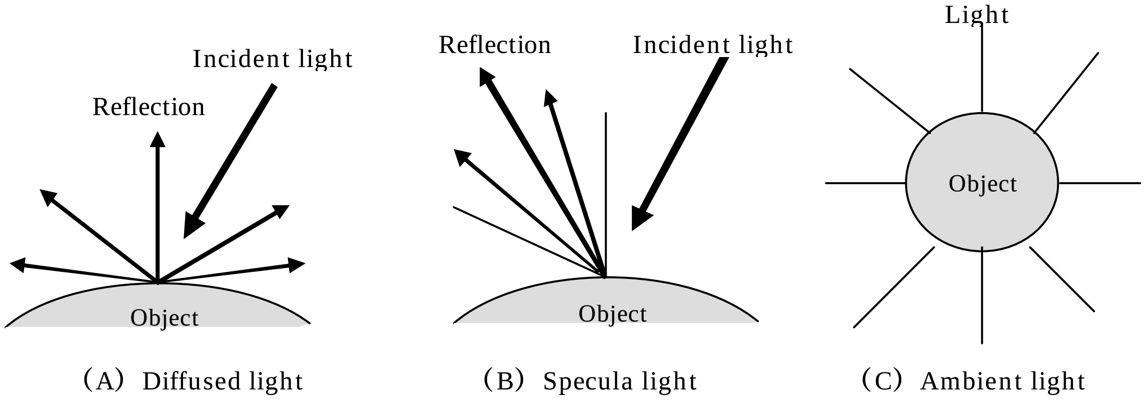


Figure2.7.3 Variety of reflection light

【Function to use this features】

The configuration of color and the brightness of the diffused light, the specula light, and the ambient light are performed by **glLight*** function.

2.7.4. Attenuation constant

The attenuation constant calculates the extinction ratio of light, reflected from a light. The following three attenuation constants are configured for each light. The extinction ration, which the distance from the light is D, is determined by Formula2.7.4.

- **Fixed attenuation constant**

  This parameter shows the extinction ratio, which is independent on the distance between a light and an object.

- **Linear attenuation constant**

  This parameter shows the extinction ratio, which is in proportion with the distance between a light and an object.

- **Quadric attenuation constant**

  This parameter shows the extinction ratio, which is in proportion with the square value of the distance between a light and an object.

$$\text{Attenuation ratio} = \frac{1}{(\text{Fixed attenuation ratio}) + (\text{Linear attenuation ratio} \times D) + (\text{Quadric attenuation ratio} \times D \times D)}$$

＜Formula2.7.4>

* D is the distance from a ligh

【Function to use this features】

The configuration of the attenuation constant of a light is performed by **glLight*** function.

2.7.5. Brightness distribution index

The brightness distribution index shows the relation of the projection angle of the spot light and the projection amount. As shown Figure2.7.5, in the spot light, the centrosphere is most brightness, and it is getting darker on the fringe. This is caused that the reflection amount is decreasing as the angle of the reflection light from the spot light is getting bigger.

If the intensity of each reflection light (diffused light, specula light, ambient light) from the spot light is “A”, the projection amount in the angle of ? direction between the projection direction is calculated by Formula2.7.5.

If the brightness distribution index is 0, the projection amount is constant regardless of the projection angle θ.

$$\begin{cases} \text{Projection amount} = A\times(\cos\theta)^{S} & (\theta \leqq \text{Cutoff}) \\ \text{Projection amount} = 0 & (\theta > \text{Cutoff}) \end{cases}$$

Formulaエラー!

* S=brightness distribution index, A=projection amount, θ=projection direction

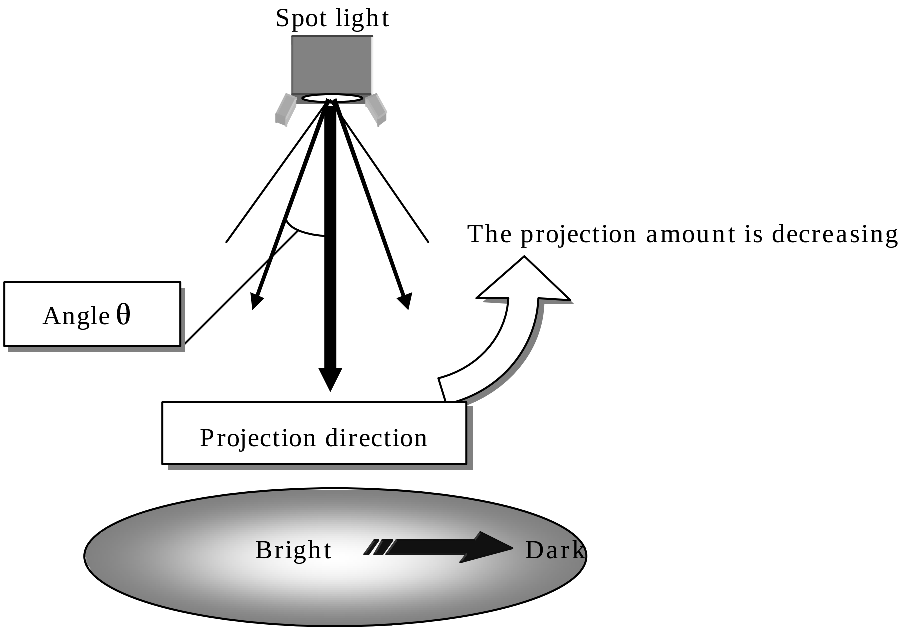


Figure2.7.5 Brightness distribution

【Function to use this features】

The configuration of the brightness distribution index of a light is performed by **glLight*** function.

2.7.6. Reflection coefficient of diffused light, specula light, ambient light

The reflection coefficient of the diffused light, the specula light, and the ambient light is the material parameter, which determines the reflection amount of the diffused light, the specula light, and the ambient light, which is projected from a light.

The configuration of each reflection coefficient is performed by each component, Red, Green, and Blue. The reflection amount of each light is determined by Formula2.7.6a, Formula2.7.6b, and Formula2.7.6c, when the angle of the incident light and the object normal line is α. The calculation of the reflection amount of each light is performed by each component, Red, Green, and Blue.

$$\begin{cases} \underset{\text{of diffused light}}{\text{Reflection amount}} = \underset{\text{of diffused light}}{\text{Intensity}} \times \cos\alpha \times \underset{\text{of diffused light}}{\text{Reflection coefficient}} & (\cos\alpha \geqq 0) \\ \text{Reflection amount of diffused light} = 0 & (\cos\alpha < 0) \end{cases}$$

＜Formulaエラー！参照元が見つかり

$$\begin{cases} \underset{\text{of specula light}}{\text{Reflection amount}} = \underset{\text{of specula light}}{\text{Intensity}} \times \underset{\text{of specula light}}{\text{Reflection coefficient}} & (\cos\alpha \geqq 0) \\ \text{Reflection amount of specula amount} = 0 & (\cos\alpha < 0) \end{cases}$$

＜Formulaエラー！参照元が見つかりませ

＊α is an angle between an incident light and the normal line of the face.

$$\underset{\text{of ambient light}}{\text{Reflection amount}} = \underset{\text{of ambient light}}{\text{Intensity}} \times \underset{\text{of ambient light}}{\text{Reflection coefficient}}$$

＜Formula2.7.6c＞

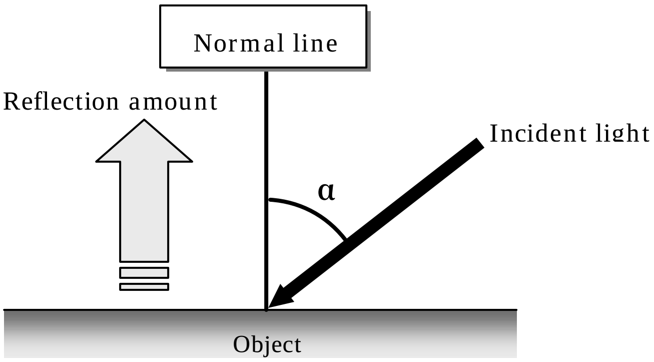


Figure2.7.6 Incident light and reflection amount

【Function to use this features】

The configuration of the reflection coefficient of the diffused light, the specula light, and the ambient light is performed by **glMaterial*** function.

2.7.7. Emitted light

The emitted light is the light emitted by an object itself. It configures each component of Red, Green, and Blue. As the emitted light is thrown out in all directions, it does not illuminate objects around.

【Function to use this features】

The emitted brightness of the emitted light is performed by **glMaterial*** function.

2.7.8. Specula brightness distribution index

The specula brightness distribution index shows the relation of the reflection direction and the reflection amount, when the specula light reflects on the object. As the specula light reflects in directivity on the object surface, the reflection amount is varied depended on the reflection direction. As shown in Figure2.7.8, the reflection amount is the maximum in the incident direction and its symmetric direction over the normal line of the face, and it is decreasing as it is getting further from the direction. The reflection amount shown in previous Formula2.7.6b is the reflection amount of the maximum reflection direction in Figure2.7.8.

If the specula brightness distribution index is "S", and the reflection amount of the maximum reflection direction is "A", the reflection amount in the direction of the angle β of the maximum reflection direction is calculated by Formula2.7.8. If the specula brightness distribution index is 0, the decrease of the reflection amount in the reflection direction is none.

$$\text{Reflection amount of specula light} = A \times \left(\cos \frac{\beta}{2}\right)^{S}$$

＜Formulaエラー!

* S=Specula brightness distribution index

A=Reflection amount in the maximum reflection direction

ß=Angle of the maximum reflection direction

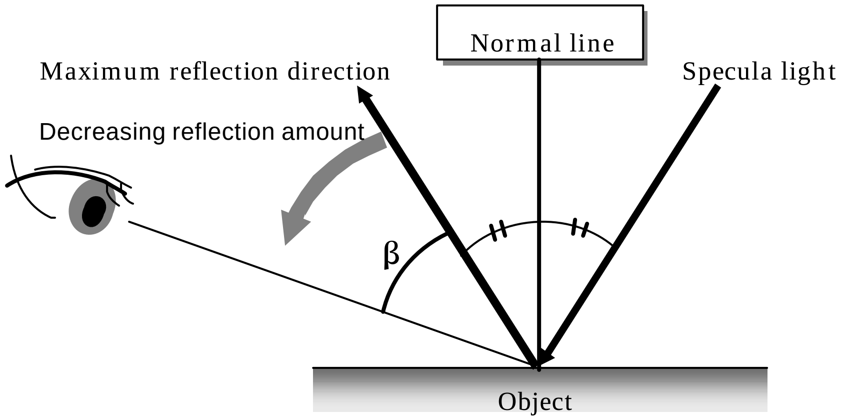


Figure2.7.8 Specula brightness

【Function to use this features】

The configuration of the specula brightness distribution index is performed by **glMaterial*** function.

2.7.9. Reflection direction of the specula light

As the specula light is reflected on the object surface in directivity, the reflection amount is varied depended on the reflection direction. There are two type of calculation method on the reflection direction of the specula light, and it is selectable.

One method is that the vector A is the reflection direction as shown in Figure2.7.9, and another is the vector B is the reflection direction as shown in same figure.

The vector A is the vector heading from the reflection point to the eye view. If it uses the vector A, the reflection amount is calculated depended on the distance between the eye view and the object, and the direction.

The vector B is the reversed direction vector of the eye view through the reflection point. If it uses the vector B, the reflection amount is calculated independed on the distance between the eye view and the object. Therefore, it is not precision compared with A, but the computational effort is reduced.

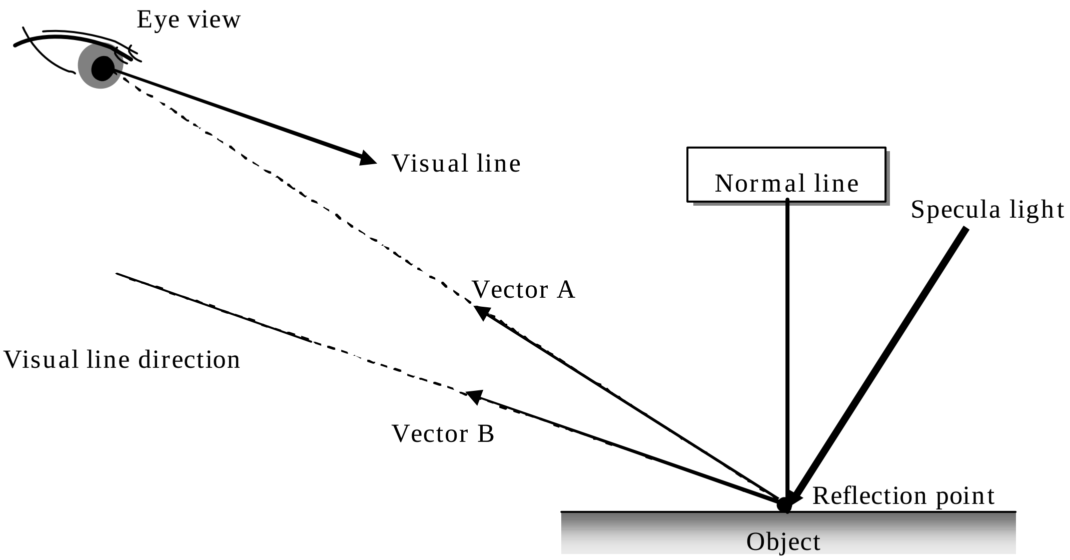


Figure2.7.9 Specula light and reflection direction

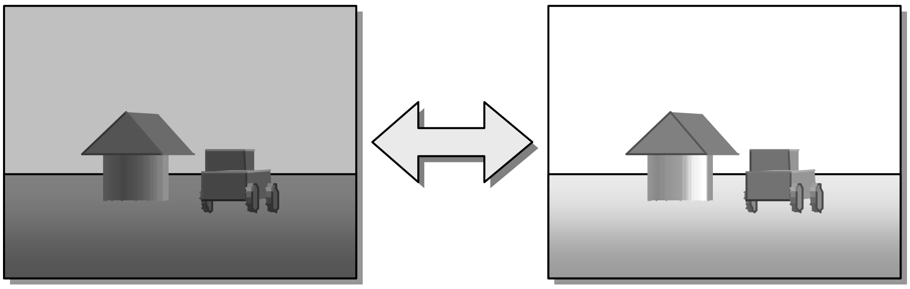
【Function to use this features】

The configuration of the reflection direction of the specula light is performed by **glLightModel*** function.

2.7.10. Ambient light of the full view

The ambient light of the full view is used to configure the brightness of the scene as shown in Figure2.7.10. As the ambient light of the full view is same characteristic as the ambient light of the light, it is independent on Valid/ Invalid of the light.

The brightness of the ambient light of the full view is configured by the value of Red, Green, and Blue.



(A)In the case of the ambient light is dark

(B)In case of the ambient light is bright

Figure2.7.10 Ambient light of the full view

【Function to use this features】

The configuration of the ambient light of the full view is performed by **glLightModel*** function.

2.7.11. Material used in the lighting calculation of the rear face

The different material could be configured for the front face and the rear face of the object. Also, in the lighting calculation of the rear face, it could select whether it uses the material of the rear face or the front face.

The front face and the rear face of the object

【Function to use this features】

The selection of the material used in the lighting calculation of the rear face is performed by **glLightModel*** function.

2.8. Texture mapping

The texture mapping is the function, which attaches a bit-mapped image on the surface of the drawing primitive. This operation is called the mapping.

It explains the details in the following.

2.8.1. Usage of the texture mapping

It specifies the texture coordinate in order to use the texture mapping, before it specifies each apex coordinate of the drawing primitive. The texture coordinate is the coordinate for specifying the texel position. The texel is the pixel, which makes up the texture image (It is a bit-mapped image used in the texture mapping.)

The texture coordinate expresses in (s, t).

Figure2.8.1 shows the example of the texture mapping.

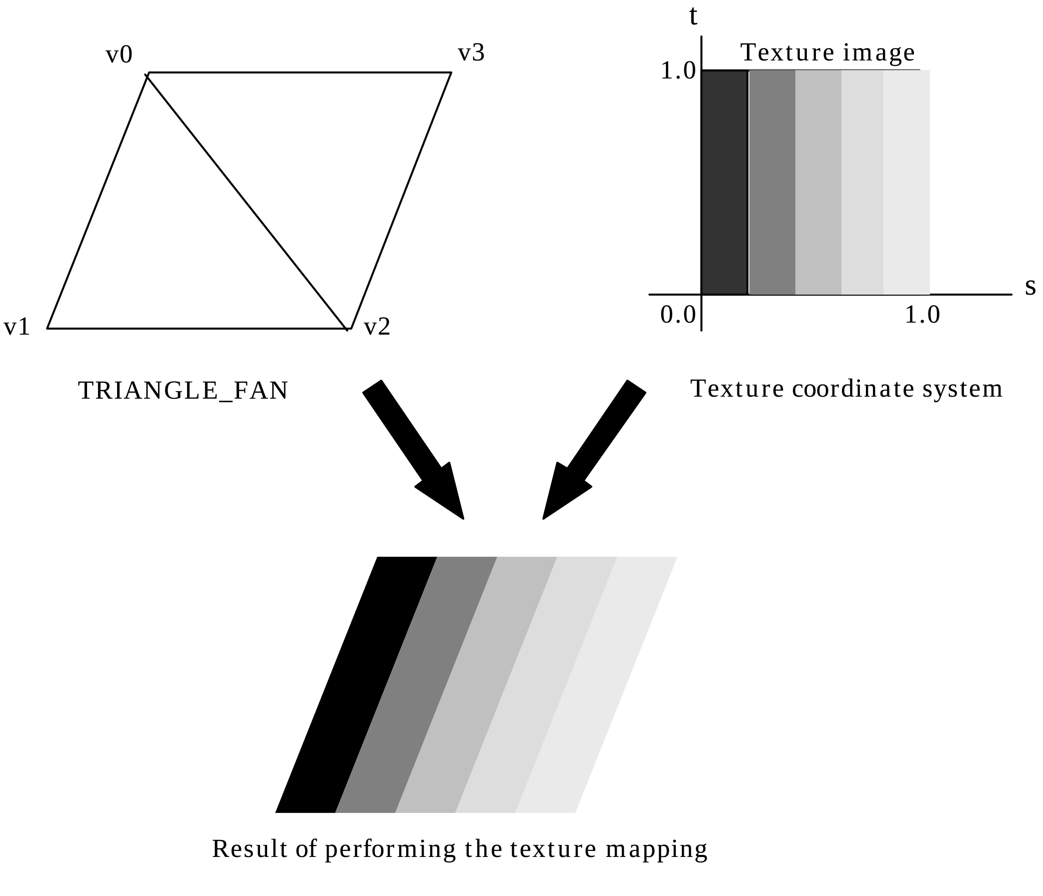

v0
v3
v1
v2
TRIANGLE_FAN
t
Texture image
1.0
s
0.0
1.0
Texture coordinate system
Result of performing the texture mapping

Figure2.8.1 Texture mapping

The physical relationship of the texture coordinate (s, t) and the texture image is as shown in Figure2.8.1, that (0.0, 0.0) is lower left, (1.0, 0.0) is lower right, (0.0, 1.0) is upper left, and (1.0, 1.0) is upper right.

In the example of the texture mapping in Figure2.8.1, the apex of the drawing primitive v0~v3 responds to each texture coordinate (0.0, 1.0), (0.0, 0.0), (1.0, 0.0), and (1.0, 1.0).

The texture coordinate is independent on the shape of the drawing primitive and the size, and it could specify in any position. If it specifies the outside of the image (above 1.0 coordinate), the wrapping is performed. It explains in *2.8.4 Texture wrapping* for this. Also, if it specifies the negative coordinate, the reversed image is mapped. If s-coordinate is the negative, it flips horizontally. If t-coordinate is the negative, it flips vertically.

【Functions to use this features】

In performing the texture mapping, it loads the texture image to the graphics memory or the internal texture image by using **glTexImage2D** in advance, and it turns the texture mapping *Invalid* using **glEnable** function.

The configuration of the texture image is performed by **glTexCoord2*** function before it specifies the apex coordinate by **flVertex3*** function. If there are three drawing primitives, it calls **glTexCoord2*** function, and then it calls **glVertex3*** function. It repeats to call those functions three times.

Notice）

・The available size of the texture image and value of the texture coordinate is differing depended on the graphics controller. (refer to *Appendix A The comparative chart of the graphics controller*)

・There is a graphics controller, which does not mount the internal texture memory. (refer to *Appendix A The comparative chart of the graphics controller*)

2.8.2. Color format of the texture

The color format of the texture image and the border color (refer to *2.8.4 Texture wrapping*) is the format, which the graphics controller defines in Figure2.8.2. The border color is specified by RGBA mode, and it is transformed to the format shown in Figure2.8.2 during the drawing.

“A” is an alpha bit. The usage of an alpha bit is explained in *2.86 Texture Blending* and *2.8.7 Texture alpha blending.*

Bit	15	14 10	5	0
	A	R	G	B

Figure2.8.2 Color format of the texture

2.8.3. Texture filter

The texture filter is the function, which interpolates the texel color.

As the texture mapping is performed with the texture image transforming corresponding with the area and the shape of the texture image, each pixel for drawing and the texel might not respond to one to one. For example, if it performs the mapping with the texture image enlarging, the number of the pixel for drawing is greater than the texel. Therefore, it creates new texels by interpolating the texel color in the texture mapping.

The following methods are selectable for the texture filter.

- **NEAREST**

It uses the texel, which is closest to the texture coordinate of the pixel for drawing. (Refer to Figure2.8.2a)

- **LINEAR**

It makes a neutral color using four texel neighborhood of the texture coordinate of the pixel for drawing. (Refer to Figure2.8.2b)

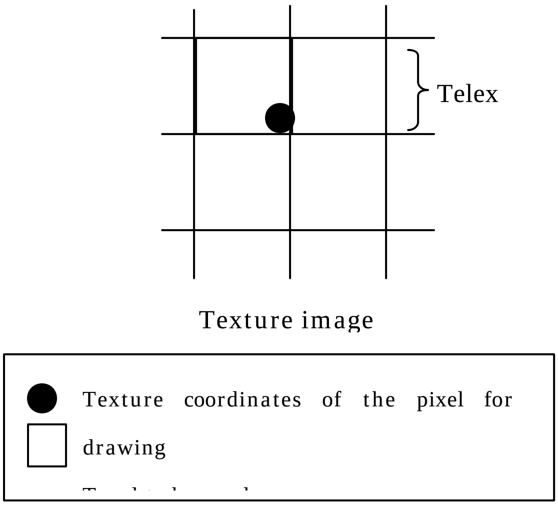


Figure2.8.3a NEAREST filter

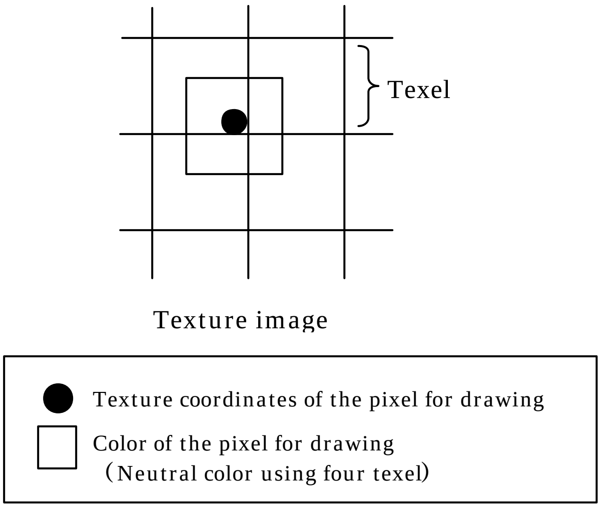

Texel
Texture image
● Texture coordinates of the pixel for drawing
□ Color of the pixel for drawing
（Neutral color using four texel)

Figure2.8.3b LINEAR filter

【Function to use this features】

The selection of the texture filter method is performed by **glTexParameter*** function.

2.8.4. Texture wrapping

The texture wrapping shows the processing, which the texture image is larger than the texture coordinate.

One of the following is selectable for each s-coordinate direction and t-coordinate direction in the texture wrapping. Figure2.8.4 shows the performing image of each texture wrapping.

■ **REPEAT**

It repeats to map the texture image.

■ **CLAMP**

If each of s-coordinate, t-coordinate is the negative, it becomes 0.0. When it is above 1.0, it performs the saturated process. As the result, it repeats to draw the pixel of the texture image edge for the area, which is outside of the texture image size.

■ **BORDER COLOR**

It draws the area, which is outside of the texture image size, with the color specified in advance. This color is called the border color.

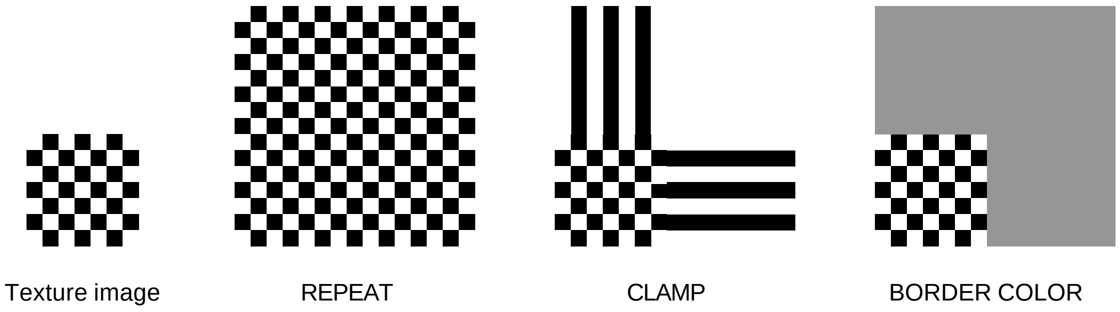


Figure2.8.4 Texture wrapping

【Function to use this features】

The specification of the texture wrapping is performed by **glTexParameter*** function.

2.8.5. Texture correction

It is the function of the perspective correction, which performs the texture mapping that the object of the perspective projection does not have any deformation.

The texture correction is selectable, either Valid/ Invalid.

Figure2.8.5 shows the performing image of the texture correction.

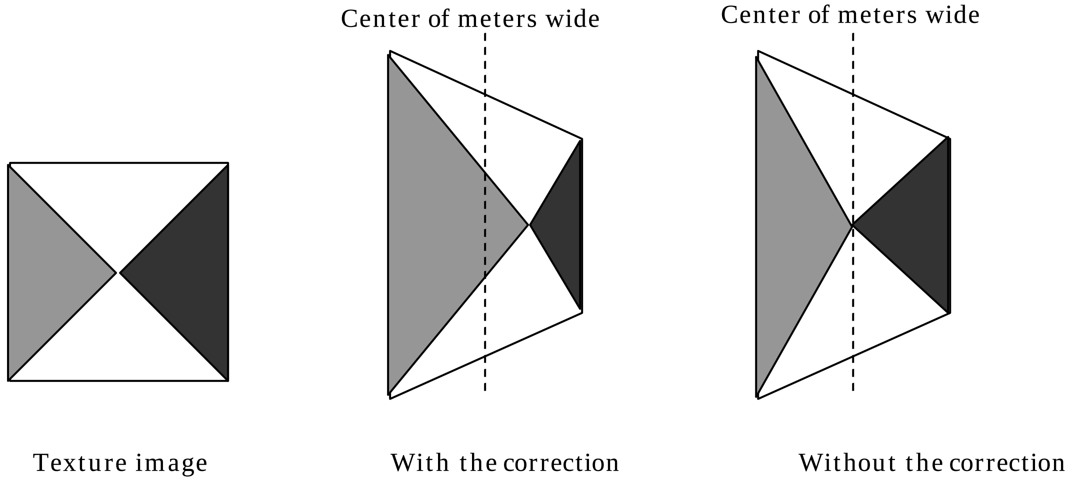


Figure2.8.5 Texture correction

【Function to use this features】

The specification of the texture correction is performed by **glEnable** function.

2.8.6. Texture blending mode

The texture blending mode shows the method, which determines color of each pixel in the texture mapping. The following is selectable in the texture blending mode.

- **DECAL**

  The texel color is color of each pixel.

- **MODULATE**

  It blends the shading color and the texel color.

- **STENCIL**

  If the alpha bit (MSB of the texel color) is 1, it uses the texel color. If it is 0, it uses the shading color.

【Function to use this features】

The texture blending mode is specified by **glTexEnv*** function.

2.8.7. Texture alpha blending

The texture alpha blending is the function that blends the pixel color and the corresponding pixel color inside the drawing frame, when it draws each pixel determined by the texture mapping.

The following methods are selectable.

- **ALL**

  It blends the consecutive color of the texture mapping and colors of the drawing frame.

- **STENCIL**

  If an alpha bit of the texel color is 1, it draws with the consecutive color of the texture mapping. If it is 0, it does not draw.

- **STENCILALPHA**

  If an alpha bit of the texel color is 1, it draws with colors blending the consecutive color of the texture mapping and colors of the drawing frame. If it is 0, it does not draw.

【Function to use this features】

The method of the texture alpha blending is specified by **glTexEnv*** function.

2.9. Depth test

The depth test is the shading method used the depth buffer (Z buffer).

The depth buffer is the memory area, which is acquired by the drawing frame and the isotopic frame (the number of pixel is same in vertical and horizontal), and z-value, which is correspond to each pixel of inside the drawing frame, is written.

The depth test compares the z-value of the pixel to draw and the z-value written in the depth buffer, and determines whether it draws or not. The method of the determination is selectable from various types.

【Function to use this features】

In order to use the depth buffer, it creates the depth buffer by **glCreateBuffer** function, and it turns the depth buffer *Valid* by **glEnable** function.

Also, **glDepthFunc** function is used for the selection of the determination method.

2.10. Special process of the line

It is possible to perform the following special process, in case of drawing a line (**GL_LINES**), a consecutive line (**GL_LINE_STRIP**), and a consecutive looped line (**GL_LINE_LOOP**).

- **Anti-aliasing**

  The shaggy hatched line is drawn smoothly by blending with the pixel color of the drawing frame.

- **Configuration of the line width**

  It configures the line width by a pixel unit. The configuration range is 1~32 pixels.

- **Broken line**

  Various broken lines are drawn by the configuration of the broken line pattern. Also, it creates the wide broken line with the combination of the configuration of the line width.

【Functions to use this features】

In order to use an anti-aliasing and the broken line, it turns each function *Valid* by **glEnable** function.

The configuration of the line width is performed by **glLineWidth** function.

The configuration of the broken line is performed by **glLineStipple** function.

3. Allocation of the graphics memory

The graphics memory is connected to the graphics controller.

The graphics controller uses the graphics memory for the various processes on the drawing.

In this chapter, it explains the usage and the allocation of the graphics memory.

3.1. Area acquired for the graphics memory

The graphics controller refers to the graphics memory, and the graphics memory is used for the drawing frame, the depth buffer, and so on.

Before using 3DGL core API, the application developer needs to determine the allocation of the graphics memory for each application.

In the following, it explains items, which are allocated on the graphics memory. Each item could be allocated on any address.

- **Drawing frame**

  The graphics controller draws on the area, the drawing frame. It is configured by 32 pixels unit, and the maximum is 4096x4096 pixels. It is required 2 byte for 1 pixel.

  The configuration of the drawing frame is performed by **glDrawDimension** function.

- **Depth buffer（Z buffer）**

  This is area used for the shading by Z buffer method. It is same form as the drawing frame, and it is required 2 byte for 1 pixel.

  The configuration of the depth buffer is performed by **glCreateBuffer** function.

- **Texture buffer**

  It is the area, which stores the texture image.

  Total size of the texture images is required.

  The configuration of the texture buffer is performed by **glTexImage2D** function.

- **DL buffer（in using the local display list transmitting）**

  It is the area, which stores the display list.

  Refer to *MB86290 Series Graphics Driver V02 User Manual* for details.

4.Programming

In this chapter, it explains the basic procedure of the application programming used 3DGL core API.

4.1. Necessary information

It describes the necessary information on the programming.

4.1.1. Include the header file

Include the following header files in the top of the source file.

vgl.h	Header files for 3D graphics library core API
gdc.h	Header file for MB86290 Series Graphics Driver

4.1.2. Create the system dependency function

Create the following three system dependency functions following the interface, which is defined by the graphics driver. Refer to *MB86290 Series Graphics Driver V02 User Manual* for details of the system dependency function.

- **GdcFlushDisplayList** function
- **XGdcSwitchDLBuf** function
- **GdcWait** function

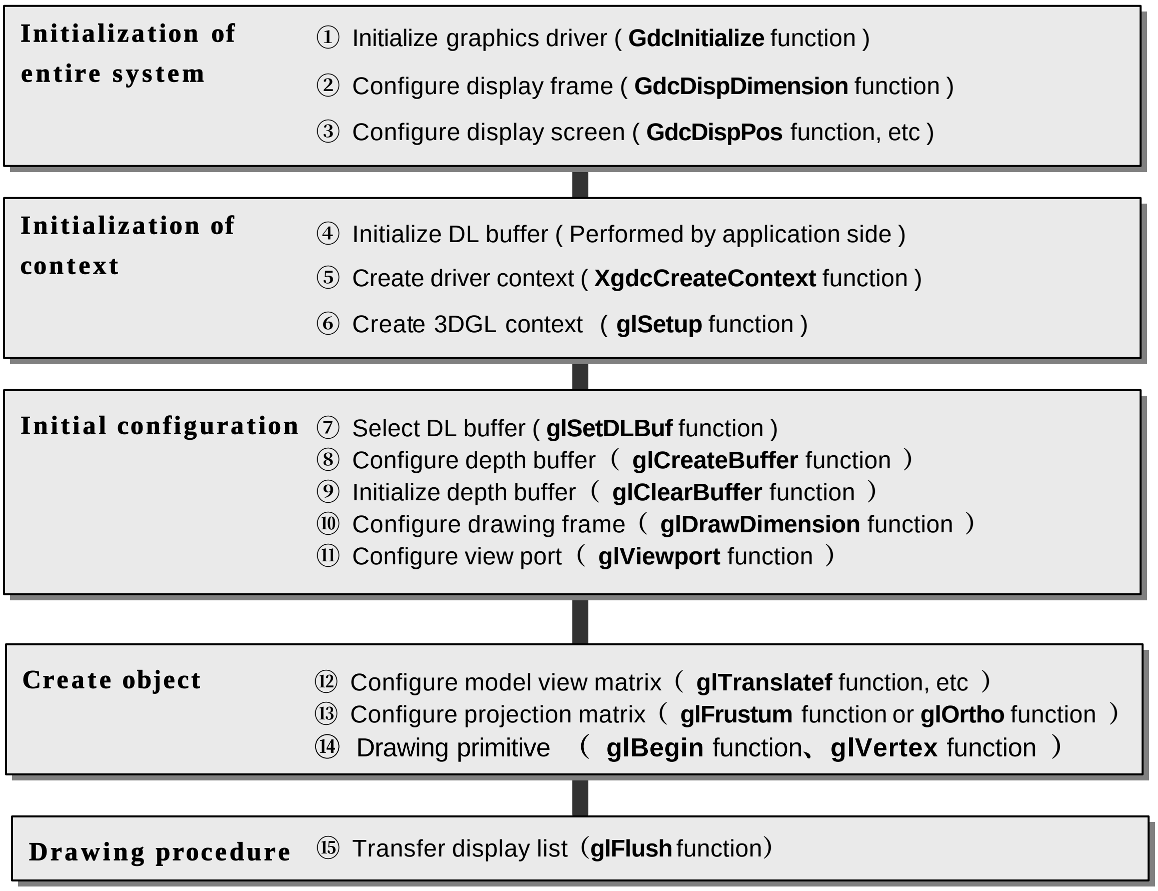
4.1.3. Reservation of various buffers

Reserve the following buffers. Refer to *3.1 Area acquired for the graphics memory* for each content.

- Drawing frame
- DL buffer
- Depth buffer
- Texture buffer

4.2. Basic process procedure

The basic process procedure used 3DGL core API is as shown in Figure4.2. Refer to the accessories program for an example.



Initialization of entire system

① Initialize graphics driver (**GdcInitialize** function)
② Configure display frame (**GdcDispDimension** function)
③ Configure display screen (**GdcDispPos** function, etc)

Initialization of context

④ Initialize DL buffer (Performed by application side)
⑤ Create driver context (**XgdcCreateContext** function)
⑥ Create 3DGL context (**glSetup** function)

Initial configuration

⑦ Select DL buffer (**glSetDLBuf** function)
⑧ Configure depth buffer （ **glCreateBuffer** function ）
⑨ Initialize depth buffer （ **glClearBuffer** function ）
⑩ Configure drawing frame （ **glDrawDimension** function ）
⑪ Configure view port （ **glViewport** function ）

Create object

⑫ Configure model view matrix （ **glTranslatef** function, etc ）
⑬ Configure projection matrix （ **glFrustum** function or **glOrtho** function ）
⑭ Drawing primitive （ **glBegin** function、**glVertex** function ）

Drawing procedure

⑮ Transfer display list（**glFlush** function）

Figure4.2 Process procedure of program

4.3. Multi task programming

It explains the necessary information in using 3DGL core API by the multi tasks.

4.3.1. Create context

Create the driver context and 3DGL context for each task. If it happens the task switch during performing 3DGL core API because of using different context for each task, the independency of the display list for each task is guaranteed.

4.3.2. Recovery of drawing property

For each task, after performing **glSetDLBuf** function, perform **XgdcRestoreAttr** function before using 3DGL core API. **XGdcRestoreAttr** function creates the display list, which sets the drawing property of the graphics controller up as same condition as the drawing property, which is stored in the driver context. This display list is added to top of the display list, which is performed in the next place by the graphics controller, and then this enables correctly draw even if another task changes the drawing property of the graphics controller.

The property for the recovery, which is specifies by **XGdcRestoreAttr** function are the common drawing property (**GDC_RESTORE_COMMON**) and the geometry drawing property (**GDC_RESTORE_GEOMETRY**). Still, there is no property for the recovery after it performed first **glSetDLBuf** function before using 3DGL core API, therefore it is not necessary to perform **XGdcRestoreAttr** function.

It shows the described example in the following.

```
XGdcCreateContext(&gdc_ctx, dlbufinfo);
glSetup(&gl_ctx, &gdc_ctx);

glSetDLBuf(&gl_ctx, 0);
/* At this point, XGdcRestoreAttr function is not necessary */
glCreateBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT, adrs);
glClearBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT);

glMatrixMode(&gl_ctx, GL_MODELVIEW);
glLoadIdentity(&gl_ctx);

glMatrixMode(&gl_ctx, GL_PROJECTION);
glLoadIdentity(&gl_ctx);

glColor3f(&gl_ctx, …);
glBegin(&gl_ctx, GL_LINES);
    :
glEnd(&gl_ctx);
```

```
glFlush(&gl_ctx);           /* First transfer of the display list */

glSetDLBuf(&gl_ctx, 1);
XGdcRestoreAttr(&gdc_ctx, GDC_RESTORE_COMMON| GDC_RESTORE_GEOMETRY);
glClearBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT);

glColor3f(&gl_ctx, …);
glBegin(&gl_ctx, GL_LINES);
    :
glEnd(&gl_ctx);
glFlush(&gl_ctx);           /* Second transfer of the display list */
```

4.3.3. Exclusive access control of transferring the display list

glFlush function and **glFlushEx** function are not performed concurrently in the multi task. If it starts to transfer the display list, perform the exclusive access control until the end of this transfer.

One example of the exclusive access control is to use the semaphore inside the system dependency function **GdcFlushDisplayList** function. It is possible to use the exclusive access control by acquiring the resource (resource=graphics controller) by this function and releasing the resource by the drawing terminate interrupt of the graphics controller (using **glInterrupt** function). However, in this process, if it is waiting to release the source, it can not perform the following the process. Also, if multi tasks are waiting to release the source, next task is determined by the priority of the execution, not the order of the waiting list. (Specially, the task order in the same priority might not be controlled.)

To solve this problem, the queue for the transfer request is required, not the semaphore. It shows an example in the following. In this organization, the transfer of the display list is performed in the block by the special task (the transfer task), not each task. Each task (drawing task 1~3), which performs the play back process of the scene, asks the transfer for the transfer task by registering the queue for the transfer request of the display list instead of performing **glFlush** function. The transfer task surveilles the queue, and if the transfer request occurs, it picks it up in the order and processes it.

The execution order of the drawing is determined by the registering order to the queue by using the queue as just described. Also, it can continue the following process, after each drawing task registers the queue for the transfer request.

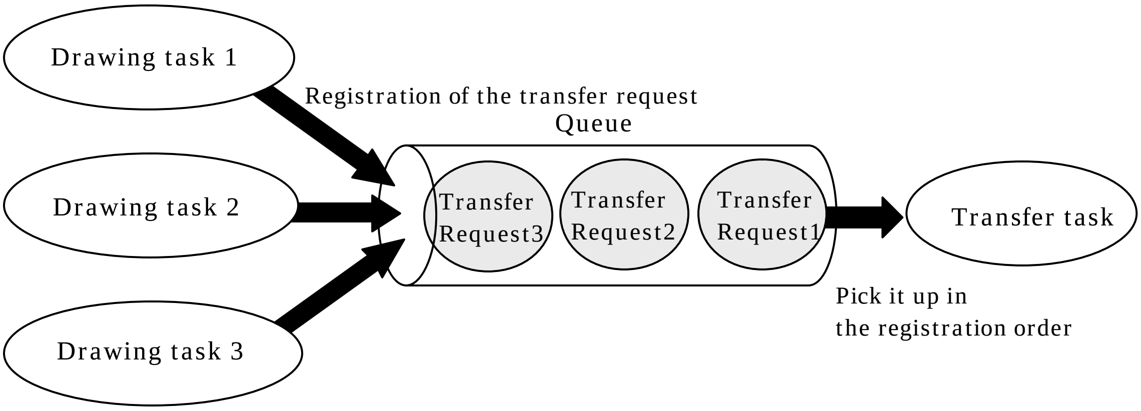


Figure4.3.3 Queue for the transfer request

4.4. Multi driver drawing functions

It explains the necessary information in using 3DGL core API and the graphics driver drawing function at the same time.

4.4.1. Create the driver context

The driver context, which is used by the drawing function of the graphics driver, and the driver context, which is used to create 3DGL context by **glSetup** function, are created separately. If it uses the same driver context, 3DGL context and the driver context become something wrong, and then 3DGL core API might not operate correctly.

4.4.2. Recovery of the drawing property

In the drawing process by 3DGL core API, after performing **glSetDLBuf** function, perform **XGdcRestoreAttr** function before using another function of 3DGL core API. Also, in the drawing process by the graphics driver too, after performing **XGdcSetDLBuf** function, perform **XGdcRestoreAttr** function before using the drawing function. **XGdcRestoreAttr** cerates the display list, which sets the drawing property of the graphics driver up as same condition as the drawing property of the driver context. This display list is added to top of the display list, which is executed in the next place, and then this enables correctly draw even if the drawing function of 3DGL core API and the graphics driver configure the different drawing property.

The properties for the recovery specified by **XGdcRestoreAttr** function are the common drawing property (**GDC_RESTORE_COMMON**) and the geometry drawing property (**GDC_RESTORE_GEOMETRY**) on 3DGL core API/ the scene drawing API. Still, it is not necessary perform **XGdcRestoreAttr** function, as it does not have the property of the recovery after executing first **glSetDLBuf** function before using 3DGL core API. It is same in performing first **XGdcSetDLBuf** function.

It shows a described example in the following.

```
XGdcCreateContext(&gdc_ctx1, dlbufinfo1);
XGdcCreateContext(&gdc_ctx2, dlbufinfo2);
glSetup(&gl_ctx, &gdc_ctx1);

/************************************
 * Drawing by 3DGL
 ************************************/
glSetDLBuf(&gl_ctx, 0);
/* XGdcRestoreAttr function is not necessary here */
glCreateBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT, adrs);
```

```
glClearBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT);

glMatrixMode(&gl_ctx, GL_MODELVIEW);
glLoadIdentity(&gl_ctx);

glMatrixMode(&gl_ctx, GL_PROJECTION);
glLoadIdentity(&gl_ctx);

glColor3f(&gl_ctx, …);
glBegin(&gl_ctx, GL_LINES);
    :
glEnd(&gl_ctx);
glFlush(&gl_ctx);

/************************************
 * Drawing by the driver function
 ************************************/
XGdcSetDLBuf(&gdc_ctx2, 0);
/* XGdcRestoreAttr function is not necessary here */
XGdcColor(&gdc_ctx2, …);
XGdcGeoPrimType(&gdc_ctx2, GDC_POLYGON);
    :
XGdcGeoPrimEnd(&gdc_ctx2);
XGdcFlush(&gdc_ctx2);

/************************************
 * Drawing by 3DGL
 ************************************/
glSetDLBuf(&gl_ctx, 1);
XGdcRestoreAttr(&gdc_ctx1, GDC_RESTORE_COMMON| GDC_RESTORE_GEOMETRY);
glClearBuffer(&gl_ctx, GL_DEPTH_BUFFER_BIT);

glColor3f(&gl_ctx, …);
glBegin(&gl_ctx, GL_LINES);
    :
glEnd(&gl_ctx);
glFlush(&gl_ctx);

/************************************
 * Drawing by the driver function
 ************************************/
XGdcSetDLBuf(&gdc_ctx2, 1);
XGdcRestoreAttr(&gdc_ctx2, GDC_RESTORE_COMMON| GDC_RESTORE_GEOMETRY);
XGdcColor(&gdc_ctx2, …);
XGdcGeoPrimType(&gdc_ctx2, GDC_POLYGON);
    :
XGdcGeoPrimEnd(&gdc_ctx2);
XGdcFlush(&gdc_ctx2);
```

5. List of 3D Graphics Library core API

In this chapter, it explains the following.

- Function list of 3DGL core API
- Symbolic constant list defined by 3DGL core API
- Data format defined by 3DGL core API

5.1. Function list

It shows the function list of 3DGL core API in the following.

Table5.1a System control function

Function name	Function
glSetup	Create the context
glRelease	Release the context
glFlush	Transfer process of the display list （current DL buffer）
glFlushEx	Transfer process of the display list （any DL buffer）
glCancelDisplayList	Cancel the display list
glVerticalSync	Create the command, which waits the vertical synchronous signal
glInterrupt	Create the interrupt command
glSetDLBuf	Select the current DL buffer
glDrawDimension	Configure the drawing frame

Table5.1b Start/ End function of the primitive

Function name	Function
glBegin	Start of the range of the primitive and the group, which belongs to it
glEnd	End of the range of the primitive and the group, which belongs to it

Table5.1c Configuration function of the shading model

Function name	Function
glShadeModel	Configuration of the shading method

Tabel5.1d Color configuration function

Function name	Function
glColor3f	Configure the current color　（RGB single precision floating point）
glColor3fv	Configure the current color array　（RGB single precision floating point）
glColor3i	Configure the current color　（RGB signed 32-bit integer）
glColor3iv	Configure the current color array　（RGB signed 32-bit integer）
glColor3ui	Configure the current color　（RGB unsigned 32-bit integer）
glColor3uiv	Configure the current color array　（RGB unsigned 32-bit integer）
glColor3ub	Configure the current color　（unsigned 8-bit integer）
glColor3ubv	Configure the current color array　（RGB unsigned 8-bit integer）
glAlphaub	Configure the current blend index　（unsigned 8-bit integer）
glAlphaf	Configure the current blend index　（single precision floating point）
glAlphai	Configure the current blend index　（signed 32-bit integer）
glAlphaui	Configure the current blend index　（unsigned 32-bit integer）
glClearColor	Configure the clear color　（RGB single precision floating point）
glBackColor	Configure the current back-ground color　（RGB unsigned 16-bit integer）

Table5.1e Apex configuration function

Function name	Function
glVertex3f	Configuration of the apex coordinate　（single precision floating point）
glVertex3fv	Configuration of the apex coordinate　（single precision floating point）
glVertex3i	Configuration of the apex coordinate　（signed 32-bit integer）
glVertex3iv	Configuration of the apex coordinate array　（signed 32-bit integer）
glNormal3f	Configuration of the normal vector
glNormal3fv	Configuration of the normal vector array

Table5.1f Matrix transformation function

Function name	Function
glFrustum	Configure the perspective projection matrix transformation
glOrtho	Configure the orthogonal projection matrix transformation
glMatrixMode	Configure the current matrix
glLoadIdentity	Replace the current matrix with the identity matrix
glPushMatrix	Push the current matrix stack
glPopMatrix	Pop the current matrix stack
gLoadMatrix	Load the matrix
glMultiMatrixf	Multiplication of the matrix
glTranslatef	Configure the translate matrix transformation
glRotatef	Configure the rotation matrix transformation
glScalef	Configure the scaling up/ down matrix transformation

Table5.1g lighting function

Function name	Function
glMaterialfv	Configure the material parameter （single precision floating point format）
glMaterialiv	Configure the material parameter （signed 32-bit integer）
glMaterialubv	Configure the material parameter （unsigned 8-bit integer）
glLightf	Configure the light parameter （single precision floating point format）
glLighti	Configure the light parameter （signed 32-bit integer）
glLightfv	Configure the light parameter （single precision floating point integer）
glLightiv	Configure the light parameter （signed 32-bit integer）
glLightubv	Configure the light parameter （unsigned 8-bit integer）
glLightModelfv	Configure the light model parameter （single precision floating point format）
glLightModeliv	Configure the light model parameter （signed 32-bit integer）

Table5.1h Line property function

Function name	Function
glLineWidth	Configure the line width
glLineStipple	Configure the broken line pattern

Table5.1I Texture configuration function

Function name	Function
glTexImage2D	Configure 2D texture image
glTexMemoryMode	Select the reference texture
glTexParameteri	Configure the texture parameter (signed 32-bit integer)
glTexEnvi	Configure the texture ambient parameter (signed 32-bit integer)
glTexParameterfv	Configure the texture parameter (single precision floating point format)
glTexParameteriv	Configure the texture parameter (signed 32-bit integer)
glTexCoord2f	Configure the current texture coordinate (single precision floating point format)
glTexCoord2fv	Configure the current texture coordinate array (signed 32-bit integer)

Table5.1j Buffer control function

Function name	Function
glClearBuffer	Clear the buffer
glCreateBuffer	Create the buffer
glDepthMask	Control the write enable of the depth buffer
glDepthFunc	Configure the value used in the depth test
glClear	Clear the view port area

Table5.1k View port transformation function

Function name	Function
glDepthRange	Configure the z-coordinate used in translating to the device coordinate
glViewport	Configure the view port coordinate

Table5.1l Enable/ Disable function

Function name	Function
glEnable	Enable the function
glDisable	Disable the function

Table5.1m shading function

Function name	Function
glFrontFace	Set the direction up as the front direction of the polygon
glCullFace	Set the direction up as the face for the shading

Table5.1n Parameter acquisition function

Function name	Function
glGetBooleanv	Acquire the parameter （truth-value）
glGetDoublev	Acquire the parameter （double precision floating point）
glGetFloatv	Acquire the parameter （single precision floating point）
glGetIntegerv	Acquire the parameter （signed 32-bit integer32）
glGetError	Acquire the error information
glGetLightfv	Acquire the light parameter （single precision floating point）
glGetLightiv	Acquire the light parameter （signed 32-bit integer）
glGetLightubv	Acquire the light parameter （unsigned 8-bit integer）
glGetMaterialfv	Acquire the material parameter （single precision floating point）
glGetMaterialiv	Acquire the material parameter （signed 32-bit integer）
glGetMaterialubv	Acquire the material parameter （unsigned 8-bit integer）
glGetTexEnviv	Acquire the texture ambient parameter
glGetTexParameterfv	Acquire the texture parameter （single precision floating point）
glGetTexParameteriv	Acquire the texture parameter （signed 32-bit integer）

Table5.1o Property save/ Recovery function

Function name	Function
glPushAttrib	Save the various property value
glPopAttrib	Recover the various property value

5.2. Symbolic constant list of 3DGL core API

It shows the symbolic constant list of 3DGL core API in the following.

Table5.2 Symbolic constant list

Symbolic constant	Meaning	Related function
GL_ALL_ATTRIB_BITS	Mask value of all properties	**glPushAttrib**
GL_ATTRIB_STACK_DEPTH	Total of property stack	**glGet***
GL_ALPHA_ALL	Texture alpha blending (ALL)	**glTexEnvi**
GL_ALPHA_STENCIL	Texture alpha blending (STENCIL)	**glTexEnvi**
GL_ALPHA_STENCIL_ALPHA	Texture alpha blending (STENCILALPHA)	**glTexEnvi**
GL_ALWAYS	Relation（always drawing）	**glDepthFunc**
GL_AMBIENT	Ambient light	**glLight*, glGetLight*, glMaterial*, glGetMaterial***
GL_AMBIENT_AND_DIFFUSE	Ambient light＆Diffused light	**glMaterial*, glGetMaterial***
GL_BACK	Rear direction	**glCullFace, glMaterial*, glGetMaterial***
GL_BLEND	Blending process	**glEnable, glGet***
GL_BORDER	Texture wrap （BORDER COLOR）	**glTexParameter***
GL_CCW	Direction of polygon （counter clockwise）	**glFrontFace**
GL_CLAMP	Texture wrap（CLAMP）	**glTexParameter***
GL_COLOR_BUFFER_BIT	Mask value of color buffer property	**glClearBuffer, glCreateBuffer, glPushAttrib**
GL_CONSTANT_ATTENUATION	Fixed attenuation constant of the light	**glLight*, glGetLight***
GL_CULL_FACE	Shading	**glEnable, glGet***
GL_CULL_FACE_MODE	Face for the shading	**glGet***
GL_CURRENT_ALPHA	Current alpha blending index	**glGet***
GL_CURRENT_BIT	Mask value of current property	**glPushAttrib**
GL_CURRENT_COLOR	Current color	**glGet***
GL_CURRENT_NORMAL	Current normal line	**glGet***
GL_CURRENT_TEXTURE_COORDS	Current texture coordinate	**glGet***

（continued）

Table5.2 Symbolic constant list（continued）

Symbolic constant	Meaning	Related function
GL_CW	Direction of polygon（clockwise）	**glFrontFace**
GL_DECAL	Texture blending （DECAL）	**glTexEnvi**
GL_DEPTH_BUFFER_BIT	Mask value of depth buffer property	**glClearBuffer, glCreateBuffer, glPushAttrib**
GL_DEPTH_FUNC	Depth test format	**glGet***
GL_DEPTH_RANGE	Depth buffer precision	**glGet***
GL_DEPTH_TEST	Depth test	**glEnable, glGet***
GL_DEPTH_WRITEMASK	Writing mask of depth buffer	**glGet***
GL_DIFFUSE	Diffused light	**glLight*, glGetLight*, glMaterial*, glGetMaterial***
GL_EMISSION	Emission brightness	**glMaterial*, glGetMaterial***
GL_ENABLE_BIT	Mask value of Valid/ Invalid of each property	**glPushAttrib**
GL_EQUAL	Relation (drawing if same)	**glDepthFunc**
GL_FALSE	False, Disable	**glGet***
GL_FLAT	Flat shading	**glShadeMode**
GL_FRONT	Front direction	**glCullFace, glMaterial*, glGetMaterial***
GL_FRONT_AND_BACK	Front/ Back direction	**glMaterial***
GL_FRONT_FACE	Polygon direction	**glGet***
GL_GEQUAL	Relation （drawing if and above）	**glDepthFunc**
GL_GREATER	Relation （drawing if greater）	**glDepthFunc**
GL_INVALID_ENUM	Error message	**glGetError**
GL_INVALID_OPERATION	Error message	**glGetError**
GL_INVALID_VALUE	Error message	**glGetError**
GL_LEQUAL	Relation （drawing if and below）	**glDepthFunc**
GL_LESS	Relation （drawing if smaller）	**glDepthFunc**
GL_LIGHT0 から GL_LIGHT7	Light number	**glLight*, glGetLight*, glEnable, glGet***
GL_LIGHTING	Lighting	**glEnable, glGet***
GL_LIGHT_BIT	Mask value of light property	**glPushAttrib**
GL_LIGHT_MODEL_AMBIENT	Ambient light of full view	**glLightModel*, glGet***

（continued）

Table5.2 Symbolic constant（continued）

Symbolic constant	Meaning	Related function
GL_LIGHT_MODEL_LOCAL_VIEWER	Calculation method of reflection angle	**glLightModel*, glGet***
GL_LIGHT_MODEL_TWO_SIDE	Lighting calculation of 1~2 face	**glLightModel*, glGet***
GL_LINEAR	Texture filter（LINEAR）	**glTexParameter***
GL_LINEAR_ATTENUATION	Linear attenuation constant of light	**glLight*, glGetLight***
GL_LINES	Line process	**glBegin**
GL_LINE_BIT	Mask value of line property	**glPushAttrib**
GL_LINE_ENDPOINT	Endpoint drawing of line	**glEnable, glGet***
GL_LINE_LOOP	Loop line process	**glBegin**
GL_LINE_SMOOTH	Anti-aliasing of line	**glEnable, glGet***
GL_LINE_STIPPLE	Broken line	**glEnable, glGet***
GL_LINE_STIPPLE_OFFSET	Offset of broken line pattern	**glEnable, glGet***
GL_LINE_STIPPLE_PATTERN	Broken line pattern	**glGet***
GL_LINE_STRIP	A set of line process	**glBegin**
GL_LINE_WIDTH	Line width	**glGet***
GL_LINE_WIDTH_GRANULARITY	Difference of line width, which is Adjacent anti-aliasing lines	**glGet***
GL_LINE_WIDTH_RANGE	Maximum width and minimum Width of anti-aliasing lines	**glGet***
GL_MATRIX_MODE	Current matrix stack	**glGet***
GL_MAX_ATTRIB_STACK_DEPTH	Maximum property stack	**glGet***
GL_MAX_LIGHTS	Maximum light number	**glGet***
GL_MAX_MODELVIEW_STACK_DEPTH	Total of maximum model view matrix stack	**glGet***
GL_MAX_PROJECTION_STACK_DEPTH	Total of maximum projection matrix stack	**glGet***
GL_MAX_TEXTURE_SIZE	Maximum texture size	**glGet***
GL_MAX_VIEWPORT_DIMS	Maximum view port size	**glGet***
GL_MODELVIEW	Model view matrix	**glMatrixMode**
GL_MODELVIEW_MATRIX	Model view matrix value	**glGet***
GL_MODELVIEW_STACK_DEPTH	Total stack of model view matrix	**glGet***
GL_MODULATE	Texture blending （MODULATE）	**glTexEnvi**
GL_NEAREST	Texture filter （NEAREST）	**glTexParameter***

Table5.2 Symbolic constant list（Continued）

Symbolic constant	Meaning	Related function
GL_NEVER	Relation （always not drawing）	**glDepthFunc**
GL_NOTEQUAL	Relation （drawing if equal）	**glDepthFunc**
GL_NORMALIZE	Normal line process	**glEnable, glGet***
GL_NO_ERROR	Error message	**glGetError**
GL_OUT_OF_MEMORY	Error message	**glGetError**
GL_PERSPECTIVE	Texture correction	**glEnable, glGet***
GL_POINTS	Point process	**glBegin**
GL_POLYGON	Polygon process	**glBegin**
GL_POLYGON_BIT	Mask value of polygon property	**glPushAttrib**
GL_POSITION	Position of the light	**glLight*, glGetLight***
GL_PROJECTION	Projection matrix	**glMatrixMode**
GL_PROJECTION_MATRIX	Projection matrix value	**glGet***
GL_PROJECTION_STACK_DEPTH	Total stack of projection matrix	**glGet***
GL_QUADRATIC_ATTENUATION	Quadric attenuation constant of the light	**glLight*, glGetLight***
GL_RGBA	Texture color	**glTexImage2D**
GL_REPEAT	Repeat	**glTexParameter***
GL_SHADE_MODEL	Shading model	**glGet***
GL_SHININESS	Mirror brightness distribution Index	**glMaterial*, glGetMaterial***
GL_SMOOTH	Smooth shading	**glShadeMode**
GL_SPECULAR	Mirror light	**glLight*, glGetLight*,** **glMaterial*, glGetMaterial***
GL_SPOT_CUTOFF	Emission angle of spot light	**glLight*, glGetLight***
GL_SPOT_DIRECION	Direction of spot light	**glLight*, glGetLight***
GL_SPOT_EXPONENT	Distribution of brightness	**glLight*, glGetLight***
GL_STACK_OVERFLOW	Error message	**glGetError**
GL_STACK_UNDERFLOW	Error message	**glGetError**
GL_STENCIL	Texture blending （STENCIL）	**glTexEnvi**

（continued）

Table5.2 Symbolic constant list（continued）

Symbolic constant	Meaning	Related function
GL_TEXTURE_2D	2D texture	**glTexImage2D, glTexParameter*, glGetTexParameter*, glEnable, glGet***
GL_TEXTURE_ALPHA_MODE	Texture alpha blending	**glTexEnvi, glGetTexEnviv**
GL_TEXTURE_BIT	Mask value of texture property	**glPushAttrib**
GL_TEXTURE_BORDER_COLOR	Texture border color	**glTexParameter*, glGetTexParameter***
GL_TEXTURE_ENV	Texture environment	**glTexEnvi, glGetTexEnviv**
GL_TEXTURE_ENV_MODE	Texture environment mode	**glTexEnvi, glGetTexEnviv**
GL_TEXTURE_EXT	Texture buffer type (graphics memory)	**glTexEnvi, glGetTexEnviv glTexImage2D**
GL_TEXTURE_FILTER	Texture filter	**glTexParameter*, glGetTexParameter***
GL_TEXTURE_INT	Texture buffer type （internal texture memory）	**glTexEnvi, glGetTexEnviv glTexImage2D**
GL_TEXTURE_MEMORY_MODE	Texture reference memory mode	**glTexEnvi, glGetTexEnviv**
GL_TEXTURE_WRAP_S	Wrap of texture s-coordinate	**glTexParameter*, glGetTexParameter***
GL_TEXTURE_WRAP_T	Wrap of texture t-coordinate	**glTexParameter*, glGetTexParameter***
GL_TRANSFORM_BIT	Mask value of transform property	**glPushAttrib**
GL_TRIANGLES	Triangle	**glBegin**
GL_TRIANGLE_FAN	Fan-shaped triangle	**glBegin**
GL_TRIANGLE_STRIP	Extensive form triangle	**glBegin**
GL_TRUE	Truth、Enable	**glGet***
GL_USHORT_1_5_5_5	16-bit color （texel format）	**glTexImage2D**
GL_VIEWPORT	Device coordinate of view port	**glGet***
GL_VIEWPORT_BIT	Mask value of view port property	**glPushAttrib**

5.3. Data format

It shows the data format used in 3DGL core API in Table5.3.

Table5.3 Data format

Letter(*1)	Data format	C language format	Data type name
b	Signed 8-bit integer	char	GLbyte
s	Signed 16-bit integer	short	GLshort
i	Signed 32-bit integer	int	GLint, GLsizei
f	Single precision floating point	float	GLfloat, GLclampf
d	Double precision floating point	double	GLdouble, GLclampd
ub	Unsigned 8-bit integer	unsigned char	GLubyte, GLboolean
us	Unsigned 16-bit integer	unsigned short	GLushort
ui	Unsigned 32-bit integer	unsigned int	GLuint, GLenum, GLbitfield
		void	GLvoid

*1 It is letters, which show the data type used in each function of 3DGL core API.

6. 3D Graphics Library core API reference

In this chapter, it explains each function of the interface of 3DGL core API.

It explains each function in the following format.

【Interface】

Declare the proto type of the function

【Argument】

Parameter name　Description of parameter

:

【Returned value】

Content of returned value

【Description】

Description of the functional capability of the function

【Related function】

Related function in use

【Parameter acquisition】

It is the explanation in case that the related parameter could be acquired by **glGet*** function.

6.1. System control

6.1.1. glSetup Initialization of 3D Graphics Library

【Interface】

void **glSetup**(GL_CTX **ctx*, GDC_CTX **gdc*)

【Argument】

ctx	Pointer to 3DGL context
gdc	Pointer to the driver context

【Returned value】

None

【Description】

It creates the context of 3DGL core API, and initializes it.

Specify the area, which stores 3DGL context for *ctx.* Acquire this area in the application.

Specify the pointer to the context of the graphics driver for *gdc*. Before using this function, the context of the graphics driver is created by **XGdcCreateContext** function.

In addition, in order to distinguish each context of 3DGL core API and the graphics driver, it calls the context of 3DGL core API *3DGL context* and it calls the context of the graphics driver *driver context*.

Before using each function of 3DGL core API, perform **glSetup** function once. However, if it uses 3DGL core API by multi tasks under the multi task environment, perform this function by each task and create 3DGL context individually. In this case, it is necessary to create the driver context for each task.

【Related function】

glRelease function

6.1.2. glRelease Release the context

【Interface】

GLint **glRelease** (GL_CTX *ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It release 3DGL context.

When it ends the application, be sure to perform. Also, if it use 3DGL core API again, perform **glSetup** function.

【Related function】

glSetup function

6.1.3. glFlush Transfer the current display list inside DL buffer

【**Interface**】

GLint **glFlush**(GL_CTX **ctx*)

【**Argument**】

ctx	Pointer to 3DGL context

【**Returned value**】

Size of the transferred display list (total byte)

【**Description**】

It performs the current display list inside DL buffer.

In this function, it calls **XGdcFlush** function of the graphics driver. If **XGdcFlush** function performs the transfer of the display list by DMA or the local display list transfer, it reverts without waiting the completion of the drawing process of the graphics controller. Therefore, this function also reverts without waiting the completion of the drawing process of the graphics controller. (Refer to the specification of the graphics controller on the transfer of the local display list)

The transfer method of the graphics controller is determined by the packaging method of **GdcFlushDisplayList** (system dependency function). Refer to the user manual of the graphics driver in detail.

If it transfers the display list inside any DL buffer, use **glFlushEx** function.

【**Related function**】

glFlushEx function

6.1.4. glFlushEx Transfer the display list inside any DL buffer

【**Interface**】

GLint **glFlushEx**(GL_CTX **ctx*, GLuint *bufno*)

【**Argument**】

ctx	Pointer to 3DGL context
bufno	DL buffer number

【**Returned value**】

Size of the transferred display list (total byte)

【**Description**】

It transfers the display list inside DL buffer specified by *bufno*.

In this function, it calls **XGdcFlushEx** function of the graphics driver. If **XGdcFlush** function performs the transfer of the display list by DMA or the local display list transfer, it reverts without waiting the completion of the drawing process of the graphics controller. Therefore, this function also reverts without waiting the completion of the drawing process of the graphics controller. (Refer to the specification of the graphics controller on the transfer of the local display list)

The transfer method of the graphics controller is determined by the packaging method of **GdcFlushDisplayList** (system dependency function). Refer to the user manual of the graphics driver in detail.

【**Related function**】

glFlush function

6.1.5. glCancelDisplayList Cancel the display list

【Interface】

GLint **glCancelDisplayList**(GL_CTX *ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It cancels the display list inside current DL buffer. The writing position of the display list is moved to top of the current DL buffer.

6.1.6. glVerticalSync Create the vertical synchronous waiting command

【Interface】

void **glVerticalSync**(GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

None

【Description】

It adds the command, which waits the vertical interval reference signal, to last of the display list created thus far.

In this command, the execution of the display list synchronizes the vertical interval reference signal. Therefore, the display list, which is created right before calling this function, is performed by the graphics controller with synchronizing the vertical interval reference signal.

6.1.7. glInterrupt Create the interrupt command

【**Interface**】

void **glInterrupt**(GL_CTX **ctx*)

【**Argument**】

ctx	Pointer to 3DGL context

【**Returned value**】

None

【**Description**】

It adds the command, which generates the drawing terminate interrupt to the current DL buffer.

6.1.8. glSetDLBuf Select the current DL buffer

【Interface】

GLint **glSetDLBuf** (GL_CTX **ctx*, GLuint *bufno*)

【Argument】

ctx	Pointer to 3DGL context
bufno	DL buffer number

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It sets DL buffer specified by *bufNo* up as the current DL buffer.

It cancels the display list, which is stored into DL buffer specified by *bufNo* on account of performing this function.

6.1.9. glDrawDimension Set the drawing frame

【Interface】

GLint **glDrawDimension** (GL_CTX **ctx*, GLubyte *cmode*, GLuint *dadrs*,
GLushort *dw*, GLushort *dh*)

【Argument】

ctx	Pointer to 3DGL context
cmode	Color mode（It can specify only **GDC_16BPP_FORMAT**）
dadrs	Drawing frame original address
dw	Drawign frame width（total pixel）
dh	Drawing frame height（total line）

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It sets the drawing frame.

Specify the drawing frame original address as the offset value from top of the graphics memory.

6.2.Primitive

6.2.1. glBegin Start the primitive process

【**Interface**】

void **glBegin**(GL_CTX **ctx*, GLenum *mode*)

【**Argument**】

ctx	Pointer to 3DGL context
mode	Primitive classification

【**Returned value**】

None

【**Description**】

It specifies the primitive to draw, and declares to start the drawing process.

Specify one of Table6.2.1 (next page) for *mode*.

In the programming, it specifies the apex coordinate of the primitive and the texture coordinate between the calling of **glBegin** function and **glEnc** function. These processes are performed using another 3DGL API function. (It describes later the function, which is capable of calling between **glBegin** function and **glEnd** function.)

It is necessary to call both of **glBegin** function and **glEnd** function together. If it performs **glBegin** function without performing **glEnd** function, it becomes an error.

The functions, which are capable of calling between **glBegin** function and **glEnd** function, are the following. Another function must be performed before **glBegin** function.

- **glColor3***function
- **glNormal3***function
- **glTexCoord2***function
- **glVertex3***function

If it draws with specifying the line primitive (**GL_LINES**, **GL_LINE_STRIP**, **GL_LINE_LOOP**), it can not configure colors of each apex. If it configures colors of each apex, the primitive is drawn with colors, which is configures last.

【**Related function**】

glEndfunction, **glColor3***function, **glNormal3***function, **glTexCoord2***function, **glVertex3***function

Table6.2.1 Symbolic constant, which specify the primitive

Symbolic constant	Meaning
GL_POINTS	It draws a point on the specified apex coordinate. If multi apex coordinates are specified, it draws a point on each coordinates.
GL_LINES	It draws a line, which connects two specified apex coordinates. The apex coordinate is specified in multiples of two. If more than four coordinates are specified, each of two coordinates is connected by a line. (Those lines, which connect each two coordinates, are independent.)
GL_LINE_STRIP	It draws a line, which connects the specified first apex coordinate to last apex coordinate.
GL_LINE_LOOP	It draws a line, which connects the specified first apex coordinate to last apex coordinate, and then connects between last apex and first apex.
GL_TRIANGLES	It draws a triangle, which connects the specified three apex coordinates. The apex coordinate is specified in multiples of three. If more than six coordinates are specified, it draws a triangle, which connects each three coordinates. (Those triangles, which connect each three apexes, are independent.)
GL_TRIANGLE_STRIP	After drawing a triangle, which connects the specified first three apexes coordinates, it draws a triangle, which connects second, third, and next specified apex coordinate. More than three apex coordinates could be specified, and if more than four are specified, a triangle is added by above rules.
GL_TRIANGLE_FAN	After drawing a triangle, which connects the specified first three apexes coordinates, it draws a triangle, which connects first, third, and next specified apex coordinate. More than three apex coordinates could be specified, and if more than four are specified, a triangle is added by above rules.
GL_POLYGON	It draws a polygon, which connects the specified apex coordinates. A polygon must be a gibbosity.

6.2.2. glEnd End the primitive process

【Interface】

void **glEnd**(GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

None

【Description】

It declares the end of the drawing process, which is started by **glBegin** function.

【Related function】

glBegin function, **glColor3*** function, **glNormal3*** function, **glTexCoord2*** function, **glVertex3*** function

6.3. Shading・model

6.3.1. glShadeModel Configure the shading method

【Interface】

void **glShadeModel**(GL_CTX **ctx*, GLenum *mode*)

【Argument】

ctx	Pointer to 3DGL context
mode	Symbolic constant, which shows the shading method

【Returned value】

None

【Description】

It configures the shading method of the drawing primitive.

It could specify the flat shading and the smooth shading. The initial value is the flat shading.

Specify one from the following table for *mode.*

Table6.3.1 Symbolic constant, which shows the shading mode

Symbolic constant	Meaning
GL_SMOOTH	Smooth shading
GL_FLAT	Flat shading（initial value）

【Related function】

glColor3* function, **glLight*** function, **glLightModel*** function

【Acquire the parameter】

It acquires the shading method, which is selected currently, using **glGet*** function. The argument is **GL_SHADE_MODEL**.

6.4. Color

6.4.1. glColor3* Configure the current color

【Interface】

void **glColor3f**(GL_CTX *ctx*, GLfloat *red*, GLfloat *green*, GLfloat *blue*)

void **glColor3i**(GL_CTX *ctx*, GLint *red*, GLint *green*, GLint *blue*)

void **glColor3ui**(GL_CTX *ctx, GLuint *red*, GLuint *green*, GLuint *blue*)

void **glColor3ub**(GL_CTX *ctx*, GLubyte *red*, GLubyte *green*, GLubyte *blue*)

void **glColor3fv**(GL_CTX *ctx*, GLfloat *v*)

void **glColor3iv**(GL_CTX *ctx*, GLint *v*)

void **glColor3uiv**(GL_CTX *ctx*, GLuint *v*)

void **glColor3ubv**(GL_CTX *ctx*, GLubyte *v*)

【Argument】

ctx	Pointer to 3DGL context
red	Value of red
green	Value of green
blue	Value of blue
v	Pointer to an array, which stores value of red, green, and blue

【Returned value】

None

【Description】

It configures the current color.

It configures the value of red, green, blue, which is configured for the current color, for red, green, blue of **glColor3f** function, **glColor3i** function, **glColor3ui** function, **glColor3ub** function.

It specifies the area, which stores value of red, green, blue for *v* of **glColor3fv** function, **glColor3iv** function, **glColor3uiv** function, **glColor3ubv** function.

The range of the value for each red, green, blue is the following Table6.4.1 (next page) depending on the data type. In either data type, larger value is brighter, it is white when the element of each color is maximum, and it is black when that is minimum. If it specifies the smaller value than the minimum, or it specifies the larger value than the maximum, the saturated process is done, and each value becomes the minimum value and the maximum value.

glColor3* function could be specified between **glBegin** function and **glEnd** function.

Table6.4.1 Range of the color value

Data type	Range
GLfloat	[0.0, 1.0]
GLint	[0, 2^{31}-1]
GLuint	[0, 2^{32}-1]
GLubyte	[0, 255]

【**Related function**】

glBegin function, **glEnd** function

【**Acquire the parameter**】

It can acquire the current color value using **glGet*** function. The argument is **GL_CURRENT_COLOR**.

6.4.2. glAlpha* Configure the current blend index

【Interface】

void **glAlphai**(GL_CTX **ctx*, GLint *alpha*)

void **glAlphaui**(GL_CTX **ctx*, GLuint *alpha*)

void **glAlphaub**(GL_CTX **ctx*, GLubyte *alpha*)

void **glAlphaf**(GL_CTX **ctx*, GLfloat *alpha*)

【Argument】

ctx	Pointer to 3DGL context
alpha	Current blend index

【Returned value】

None

【Description】

It configures the current blend index.

The current blend index is used as the alpha blend index.

In the blend index, the minimum value is the transmissivity 100% (it does not draw), the maximum value is the transmissivity 0% (it draws with untouched color).

The range of the value of the blend index, which is used by each **glAlpha*** function is as shown in Table6.4.2. If it specifies the smaller value than the minimum, or it specifies the larger value than the maximum, the saturated process is done, and each value becomes the minimum value and the maximum value. In addition, the gradation sequence of the alpha blending is 256 steps. Therefore, in each function, the blend index is converted to range of [0, 255].

The alpha blending is enabled by **glEnable** function, and disenable by **glDisable** function.

Table6.4.2 Range of the blend index value

Function name	Range
glAlphai	[0, 2^{31}-1]
glAlphaui	[0, 2^{32}-1]
glAlphaub	[0, 255]
glAlphaf	[0.0, 1.0]

【Related function】

glEnable function, **glDisable** function

【Acquire the parameter】

It can acquire the current blend index using **glGet*** function. The argument is **GL_CURRENT_ALPHA**.

6.4.3. glClearColor Configure the clear color

【**Interface**】

void **glClearColor**(GL_CTX **ctx*, GLfloat *red*, GLfloat *green*, GLfloat *blue*)

【**Argument**】

ctx	Pointer to 3DGL context
red	Red value for the clear color in the view port area
green	Green value for the clear color in the view port area
blue	Blue value for the clear color in the view port area

【**Returned value**】

None

【**Description**】

It configures the clear color in the view port area.

Configure the value of [0.0, 1.0] range for each red, green, blue.

The initial value is all 0.0.

【**Related function**】

glClear function

【**Acquire the parameter**】

It can configure the current value using **glGet*** function. The argument is **GL_COLOR_CLEAR_VALUE**.

6.4.4. glBackColor Configure the back ground color

【Interface】

GLint **glBackColor**(GL_CTX **ctx*, GLushort *color*)

【Argument】

ctx	Pointer to 3DGL context
color	Back ground color

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It configures the current back ground color. The back ground color is used with broken lien. (Refer to *6.8.2 glLineStipple*)

Specify the back ground color in 16 bit color format of the graphics controller for *color*.

If it set 1 up as MSB of *color*, the back ground color becomes transparent color.

【Related function】

glLineStipple function

6.5. Apex

6.5.1. glVertex3* Configure the apex coordinate

【**Interface**】

void **glVertex3f**(GL_CTX *ctx*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

void **glVertex3i**(GL_CTX *ctx*, GLint *x*, GLint *y*, GLint *z*)

void **glVertex3fv**(GL_CTX *ctx*, GLfloat *v*)

void **glVertex3iv**(GL_CTX *ctx*, GLint *v*)

【**Argument**】

ctx	Pointer to 3DGL context
x	x-coordinate of the apex
y	y-coordinate of the apex
z	z-coordinate of the apex
v	Pointer to the area, which stores x,y,z-coordinate of the apex

【**Returned value**】

None

【**Description**】

It configures the apex coordinate (object coordinate), when it draws the primitive.

Configure each apex x,y,z-coordinate value for x,y,z.

Configure the area, which stores x,y,z-coordinate value for *v*.

glVertex3* function is capable of using only between **glBegin** function and **glEnd** function call.

【**Related function**】

glBegin function, **glEnd** function, **glColor3*** function, **glMaterial*** function, **glNormal3*** function, **glTexCoord2*** function

6.5.2. glNormal3* Configure the current normal vector

【Interface】

void **glNormal3f**(GL_CTX **ctx*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

void **glNormal3fv**(GL_CTX **ctx*, GLfloat **v*)

【Argument】

ctx	Pointer to 3DGL context
x	x-coordinate, which determines the current normal line
y	y-coordinate, which determines the current normal line
z	z-coordinate, which determines the current normal line
v	Pointer to the area, which stores x,y,z-coordinate, which determines the current normal line

【Returned value】

None

【Description】

It configures the current normal vector.

Configure the area, which stores the x,y,z-coordinate value in turn, which determines the normal vector for *v*. In addition, if it specify the normal vector, which is not unit vector (length=1), it specifies **GL_NORMLIZE** by **glEnable** function, and it enables the normalization function (make the unit vector) of the normal vector.

glNormal3* function is capable of using between **glBegin** function and **glEnd** function.

【Related function】

glBegin function, **glEnd** function, **glColor3*** function, **glTexCoord2*** function, **glVertex3*** function

【Acquire the parameter】

It can acquire the current normal line using **glGet*** function. The argument is **GL_CURRENT_NORML**.

6.6. Matrix transformation

6.6.1. glFrustum Specify the perspective projection matrix transformation

【Interface】

void **glFrustum**(GL_CTX *ctx*, GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdoule *far*)

【Argument】

ctx	Pointer to 3DGL context
left	Left x-coordinate on the clip face
right	Right x-coordinate on the clip face
bottom	Lower y-coordinate on the clip face
top	Upper y-coordinate on the clip face
near	z-coordinate on the front clip face
far	z-coordinate on the back clip face

【Returned value】

None

【Description】

It configures the projection matrix for the perspective projection.

Specify x-coordinate on each left, right clip face for *left, right*.

Specify y-coordinate on each upper, lower clip face for *top, bottom*.

Specify z-coordinate on each front clip face and back clip face for *near, far*. These values must be positive.

（continued to next page）

The projection matrix is represented by the following matrix formula.

$$\begin{pmatrix} \dfrac{2 \cdot near}{right - left} & 0 & m1 & 0 \\ 0 & \dfrac{2 \cdot near}{top - bottom} & m2 & 0 \\ 0 & 0 & m3 & m4 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$m1 = (right+left) / (right - left)$

$m2 = (top+bottom) / (top - bottom)$

$m3 = -(far+near) / (far - near)$

$m4 = -(2far \cdot near) / (far - near)$

【Related function】

glOrtho function, **glMatrixMode** function, **glMultMatrix** function, **glPushMatrix** function **glPopMatrix** function, **glViewport** function

【Acquire the parameter】

It can acquire the current matrix mode using **glGet*** function. The argument is **GL_MATRIX_MODE**.

6.6.2. glOrtho Specify the orthogonal projection matrix transformation

【Interface】

void **glOrtho**(GL_CTX **ctx*, GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdoule *far*)

【Argument】

ctx	Pointer to 3DGL context
left	Left x-coordinate on the clip face
right	Right x-coordinate on the clip face
bottom	Lower y-coordinate on the clip face
top	Upper y-coordinate on the clip face
near	z-coordinate on the front clip face
far	z-coordinate on the back clip face

【Returned value】

None

【Description】

It configures the projection matrix for the orthogonal projection.

Specify x-coordinate on each left, right clip face for *left, right*.

Specify y-coordinate on each upper, lower clip face for *top, bottom*.

Specify z-coordinate on each front clip face and back clip face for *near, far*.

（continued to next page）

The projection matrix is represented by the following matrix formula.

$$\begin{pmatrix} \frac{2}{\mathit{right}-\mathit{left}} & 0 & 0 & m1 \\ 0 & \frac{2}{\mathit{top}-\mathit{bottom}} & 0 & m2 \\ 0 & 0 & \frac{-2}{\mathit{far}-\mathit{near}} & m3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$m1 = -(\mathit{right}+\mathit{left}) / (\mathit{right}-\mathit{left})$

$m2 = -(\mathit{top}+\mathit{bottom}) / (\mathit{top}-\mathit{bottom})$

$m3 = -(\mathit{far}+\mathit{near}) / (\mathit{far}-\mathit{near})$

【Related function】

glFrustum function, **glMatrixMode** function, **glMultMatrix** function, **glPushMatrix** function, **glPopMatrix** function, **glViewport** function

【Acquire the parameter】

It can acquire the current matrix mode using **glGet*** function. The argument is **GL_MATRIX_MODE**.

6.6.3. glMatrixMode Specify the current matrix

【**Interface**】

void **glMatrixMode**(GL_CTX **ctx*, GLenum *mode*)

【**Argument**】

ctx	Pointer to 3DGL context
mode	Matrix stack

【**Returned value**】

None

【**Description**】

It specifies the matrix stack targeted at the matrix calculation.

The altered matrix mode is kept until it is altered again by this function.

Specify one from the following table for *mode*.

Table6.6.3 Symbolic constant, which shows the matrix mode

Symbolic constant	Meaning
GL_MODELVIEW	It specifies next matrix calculation for the model view matrix.
GL_PROJECTION	It specifies next matrix calculation for the projection matrix.

【**Related function**】

glMatrixMode function, **glPushMatrix** function, **glPopMatrix** function

【**Acquire the parameter**】

It acquires the current matrix mode using **glGet*** function. The argument is **GL_MATRIX_MODE**.

6.6.4. glLoadIdentity Specify the unit matrix

【Interface】

void **glLoadIdentity**(GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

None

【Description】

It replaces the current enable matrix (model view matrix or projection matrix) with the unit matrix.

It can alter the current matrix mode by **glMatrixMode** function.

The unit matrix is the following value. The matrix is used as the initial value.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

【Related function】

glMatrixMode function, **glMultMatrix** function, **glPushMatrix** function, **glPopMatrix** function

【Acquire the parameter】

It can acquire the current matrix mode using **glGet*** function. The argument is **GL_MATRIX_MODE**.

6.6.5. glPushMatrix Save the matrix

【Interface】

void **glPushMatrix** (GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

None

【Description】

It copies the current matrix to top of the stack, and saves it.

The maximum stack of each matrix is following.

・Model view matrix	32 steps
・Projection matrix	2 steps

【Related function】

glPushMatrix function, **glFrustum** function, **glLoadIdentity** function, **glMatrixMode** function, **glMultMatrix** function, **glOrtho** function, **glRotatef** function, **glScalef** function, **glTranslatef** function, **glViewport** function

6.6.6. glPopMatrix Recovery the matrix

【Interface】

void **glPopMatrix** (GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

None

【Description】

It pops the stack, and replaces with the current matrix.

The maximum stack of each matrix is following.

・Model view matrix 32 steps

・Projection matrix 2 steps

【Related function】

glPushMatrix function, **glFrustum** function, **glLoadIdentity** function, **glMatrixMode** function, **glMultMatrix** function, **glOrtho** function, **glRotatef** function, **glScalef** function, **glTranslatef** function, **glViewport** function

6.6.7. glLoadMatrixf Load the matrix

【Interface】

void **glLoadMatrixf**(GL_CTX **ctx*, const GLfloat **m*)

【Argument】

ctx	Pointer to 3DGL context
m	Pointer to the matrix to configure

【Returned value】

None

【Description】

It replaces the current matrix.

The current matrix is the projection matrix, which is determined by **glMatrixMode**, or the model view matrix.

Specify the current matrix to replace for *m*. Store the matrix element in the following format.

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

【Related function】

glLoadIdentity function, **glMatrixMode** function, **glMultiMatrixf** function, **glPushMatrix** function, **glPopMatrix** function

6.6.8. glMultiMatrixf Multiplication of the matrix

【Interface】

void **glMultiMatrixf**(GL_CTX **ctx*, const GLfloat **m*)

【Argument】

ctx	Pointer to 3DGL context
m	Pointer to the matrix for the multiplication

【Returned value】

None

【Description】

The current matrix multiplies the matrix specified by *m*.

The current matrix is the projection matrix, which is determined by **glMatrixMode** function or the model view matrix.

The matrix element of *m* is stored in the following format.

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

【Related function】

glLoadIdentity function, **glMatrixMode** function, **glLoadMatrixf** function, **glPushMatrix** function, **glPopMatrix** function

6.6.9. glTranslatef Specify the movement matrix transformation

【**Interface**】

void **glTranslatef**(GL_CTX **ctx*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

【**Argument**】

ctx	Pointer to 3DGL context
x	Distance of x-coordinate
y	Distance of y-coordinate
z	Distance of z-coordinate

【**Returned value**】

None

【**Description**】

It performs the matrix transformation of the movement for the current matrix.

After performing this function, all objects, which is drawn, is moved.

Specify the movement of x, y, z-coordinate for each *x, y, z*.

If it saves the coordinate, which is pre-performing this function, perform **glPushMatrix** function in advance.

【**Related function**】

glMatrixMode function, **glMultMatrix** function, **glPushMatrix** function, **glPopMatrix** function, **glRotate** function, **glScalef** function

6.6.10. glRotatef Specify the rotate matrix transformation

【Interface】

void **glRotate**f(GL_CTX **ctx*, GLfloat *angle*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

【Argument】

ctx	Pointer to 3DGL context
angle	Rotation angle
x	x-coordinate of axis vector
y	y-coordinate of axis vector
z	z-coordinate of axis vector

【Returned value】

None

【Description】

It performs the matrix transformation of the rotation for the current matrix.

After performing this function, all objects, which are drawn, rotate.

Specify the coordinate, which determines the rotation axis, for *x, y, z*. The rotation axis is a line, which connects the origin and the coordinate specified by *x, y, z*. The rotation direction is left direction, looking the rotation axis front.

If it is necessary to save the coordinate before pre-performing this function, perform **glPushMatrix** function in advance

【Related function】

glMatrixMode function, **glMultMatrix** function, **glPushMatrix** function, **glPopMatrix** function, **glScalef** function, **glTranslatef** function

6.6.11. glScalef Specify scaling up/ down matrix transformation

【**Interface**】

void **glScalef**(GL_CTX **ctx*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

【**Argument**】

ctx	Pointer to 3DGL context
x	Scaling value of x-coordinate
y	Scaling value of y-coordinate
z	Scaling value of z-coordinate

【**Returned value**】

None

【**Description**】

It performs scaling up/ down matrix transformation for the current matrix.

After performing this function, all objects, which are drawn, are scaling up/ down.

Specify the scaling value of each x-coordinate, y-coordinate, z-coordinate for *x, y, z*.

If it is necessary to save the coordinate, which is pre-performing this function, perform **glPushMatrix** function in advance.

【**Related function**】

glMatrixMode function, **glMultMatrix** function, **glPushMatrix** function, **glPopMatrix** function, **glRotatef** function, **glTranslatef** function

6.7. Lighting

6.7.1. glLight* Configure the lighting parameter

【Interface】

void **glLightf**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, const GLfloat *param*)

void **glLighti**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, const GLint *param*)

void **glLightfv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, const GLfloat **params*)

void **glLightiv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, const GLint **params*)

void **glLightubv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, const GLubyte **params*)

【Argument】

ctx	Pointer to 3DGL context
light	Identifier of the light
pname	Classification of the lighting parameter
param	Configured value of the lighting parameter
params	Pointer to the array, which stores configured value of the lighting parameter

【Returned value】

None

【Description】

It configures the lighting parameter.

Specify one of **GL_LIGHT0~GL_LIGHT7**, which are target lighting idenfier, for *light*.

Set the lighting parameter up as the symbolic constant shown in Table6.7.1 (next page) for *pname*.

Specify the configured value correspond to the classification of the lighting parameter for *param* and *params* as shown in Table6.7.1.

If it enables the lighting, it specifies **GL_LIGHTING** by **glEnable** function. If it disenables the lighting, it specifies **GL_LIGHTING** by **glDisable** function.

Also, if it enables or disenables each the light, specify the identifier of the light by this function.

【Related function】

glLightModel* function, **glMaterial*** function

【Acquire the parameter】

It can acquire the configured value of the lighting parameter using **glGetLight*** function.

Table6.7.1 Symbolic constant and configured value on lighting parameter configuration

Symbolic constant	Meaning
GL_AMBIENT	It configures the ambient light of the light. Store the configured value in order of R, G, B, A for the area specified by *params*. The initial value is (0.0, 0.0, 0.0, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0. **GL_AMBIENT** is not specified by **glLightf** function and **glLighti** function.
GL_DIFFUSE	It configures the diffused light of the light. Store the configured value in order of R, G, B, A for the area specified by *params*. The initial value is (1.0, 1.0, 1.0, 1.0) for **GL_LIGHT0** and (0.0, 0.0, 0.0, 1.0) for **GL_LIGHT1～GL_LIGHT7**. **GL_DIFFUSE** is not specified by **glLightf** function and **glLIghti** function.
GL_SPECULAR	It configures the specula light of the light. Store the configured value in order of R, G, B, A for the area specified by *params*. The initial value is (1.0, 1.0, 1.0, 1.0) for **GL_LIGHT0** and (0.0, 0.0, 0.0, 1.0) for **GL_LIGHT1～GL_LIGHT7**. The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0. **GL_SPECULAR** is not specified by **glLightf** function and **glLIghti** function.
GL_POSITION	It configures the position of the light. Store the configured value in order of x, y, z, w for the area specified by *params.* The initial value is (0, 0, 1, 0). **GL_POSITION** is not specified by **glLightf** function and **glLighti** function.

（continued）

Table6.7.1 Symbolic constant and configured value on lighting parameter configuration

Symbolic constant	Meaning
GL_SPOT_DIRECTION	It configures the projection direction of the spot light. Store the configured value in order of x, y, z for the area specified by *params.* The initial value is (0, 0, -1). **GL_SPOT_DIRECTION** is not specified by **glLight** function and **glLighti** function.
GL_SPOT_EXPONENT	It configures the brightness distribution index of the spot light. The effective area of the configured value is [0.0, 128.0], the light source is narrow down as the larger value. The initial value is 0.0. (equally distribution) If it specifies out range of [0.0, 128.0], it does not specify the value.
GL_SPOT_CUTOFF	It configures the maximum radiation angle of the light. The effective area of the configured value is [0.0, 90.0] and 180.0. 180.0 is a special angle, it shows the spot light, and the light is projected in all directions. The initial value is 180.0. If it specifies out range of [0.0, 90.0] and non-180.0, it does not configure the value.
GL_CONSTANT_ATTENUATION	It configures the fixed attenuation constant of the light. 0 and positive value are valid as the configured value, and the initial value is 1.0.
GL_LINEAR_ATTENUATION	It configures the linear attenuation constant of the light. 0 and positive value are valid as the configured value, and the initial value is 0.0.
GL_QUADRATIC_ATTENUATION	It configures the quadratic attenuation constant of the light. 0 and positive value are valid as the configured value, and the initial value is 0.0.

6.7.2. glMaterial* Configure the material parameter

【Interface】

void **glMaterialfv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, const GLfloat **param*)

void **glMaterialiv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, const GLint **param*)

void **glMaterialubv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, const GLubyte **param*)

【Argument】

ctx	Pointer to 3DGL context
face	Target face, which configures the material parameter
pname	Classification of the material parameter
param	Pointer to area, which stores the configured value of the material parameter

【Returned value】

None

【Description】

It configures the value of the material parameter.

Set one of the following table up as the triangle target face, which configures the material parameter for *face*.

Table6.7.2a Symbolic constant, shows the configuring face of the material parameter

Symbolic constant	Meaning
GL_FRONT	Front face
GL_BACK	Back face
GL_FRONT_AND_BACK	Both sides

Specify the symbolic constant of table6.7.2b (next page) up as the configuring material parameter for *pname*.

Specify the area, which stores the value corresponding to classification of the material parameter, as shown in Table6.7.2b.

【Related function】

glLight* function, **glLightModel***function

Table6.7.2b Symbolic constant and configured value on the configuration of the material parameter

Symbolic constant	Meaning
GL_AMBIENT	It configures the reflectivity of the ambient light. Store the configured value in order of R, G, B, A for the area specifying by *param*. The initial value is (0.2, 0.2, 0.2, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.
GL_DIFFUSE	It configures the reflectivity of the diffused light. Store the configured value in order of R, G, B, A for the area specifying by *param*. The initial value is (0.8, 0.8, 0.8, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.
GL_SPECULAR	It configures the reflectivity of the specula light. Store the configured value in order of R, G, B, A for the area specifying by *param*. The initial value is (0.0, 0.0, 0.0, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.
GL_EMISSION	It configures the radiance of the emission light. Store the configured value in order of R, G, B, A for the area specifying by *param*. The initial value is (0.0, 0.0, 0.0, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.
GL_SHININESS	It configures the shininess distribution index. The effective area is [0, 128]. The initial value is 0.
GL_AMBIENT_AND_DIFFUSE	It configures same value of the reflectivity for each ambient light and diffused light. Store the configured value in order of R, G, B, A for the area specifying by *param*. The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.

6.7.3. glLightModel* Configure the bright light model parameter

【Interface】

void **glLightModelfv**(GL_CTX **ctx*, GLenum *pname*, const GLfloat **param*)

void **glLightModeliv**(GL_CTX **ctx*, GLenum *pname*, const GLint **param*)

void **glLightModelubv**(GL_CTX **ctx*, GLenum *pname*, const GLubyte **param*)

【Argument】

ctx	Pointer to 3DGL context
pname	Classification of the bright light model parameter
param	Configuring value of the bright light model parameter

【Returned value】

None

【Description】

It configures the bright light model parameter.

Specify the symbolic constant of Table6.7.3 (next page) up as the configuring bright light model parameter for *pname*.

Specify the area, which stores the value corresponding to classification of the bright light model parameter, as shown in Table6.7.3.

【Related function】

glLight* function, **glMaterial*** function

【Acquire the parameter】

It can acquire the bright light model parameter using **glGet*** function. The arguments are following three types.

・**GL_LIGHT_MODEL_AMBIENT**

・**GL_LIGHT_MODEL_LOCAL_VIEWER**

・**GL_LIGHT_MODEL_TWO_SIDE**

Table6.7.3 Symbolic constant and configured value on the configuration of the bright light model parameter

Symbolic constant	Meaning
GL_LIGHT_MODEL_AMBIENT	It configures the ambient light of the full view. Store the configured value in order of R, G, B, A for the area specifying by *param*. The initial value is (0.2, 0.2, 0.2, 1.0). The configured value specified by the integer type performs the linear mapping, corresponding the minimum value is -1.0, the maximum value is 1.0.
GL_LIGHT_MODEL_LOCAL_VIEWER	It configures the calculation method of the reflection angle of the specula reflection. If it specifies 0 for the configured value, the reflection angle makes parallel to z-axis. If it specifies except 0, it calculates from an origin of the eye point coordinate. The initial value is 0.
GL_LIGHT_MODEL_TWO_SIDE	It specifies the material of the object, which is used to calculate the lighting of back direction. If it specifies 0 for the configured value, it uses the material, which is configured in front direction. If it specifies except 0, it uses the material, which is configured in back direction.

6.8. Line width・broken line

6.8.1. glLineWidth Configure the line width

【**Interface**】

void **glLineWidth**(GL_CTX **ctx*, GLfloat *width*)

【**Argument**】

ctx	Pointer to 3DGL context
width	Line width

【**Returned value**】

None

【**Description**】

It specifies the line width. The maximum value is 32.0.

If it specifies except 1.0 for the line width, the result is differ depend on valid/ invalid of the anti-aliasing.

The initial value of the line width is 1.0.

In order to enable anti-aliasing of the line, it specifies **GL_LINE_SMOOTH** by **glEnable** function.

In order to disable anti-aliasing of the line, it specifies **GL_LINE_SMOOTH** by **glDisable** function.

【**Related function**】

glEnable function, **glDisable** function, **glLineStipple** function

【**Acquire the parameter**】

It can acquire the line width using **glGet*** function. The argument is **GL_LINE_WIDTH**.

6.8.2. glLineStipple Configure the broken line pattern

【Interface】

void **glLineStipple**(GL_CTX **ctx*, GLint *factor*, GLuint *pattern*)

【Argument】

ctx	Pointer to 3DGL context
factor	Number of counterturn of each bit of the broken line pattern(can specify only 1)
pattern	Broken line pattern of 32 bit

【Returned value】

None

【Description】

It configures the drawing pattern of the broken line.

Specify the broken line pattern with 32 bit of 0, 1 for *pattern.* The drawing patter is put in order of MSB, it draws the pixel corresponding to the position of 1 with the current color, and it draws the pixel corresponding to the position of 0 with the back ground color. (Refer to below picture) The configuration of the back ground color is performed by **glBackColor** function.

Only 1 could be specified for *factor* currently. Even if another value is specified, it treats as 1.

In order to perform the broken line drawing, it is necessary to configure the broken line patter, and also enable the broken line process. In order to enable the broken line process, specify **GL_LINE_STIPPLE** by **glEnable** function. The initial configuration of the broken line process is invalid.

pattern	Drawing result
0xAAAAAAAA	●○●○●○●○●○●○●○●○●○●○●○●○●○●○●○●○
0x00FF00FF	○○○○○○○○●●●●●●●●○○○○○○○○●●●●●●●●
0x0C0F0C0F	○○○○●●○○○○○○●●●●○○○○●●○○○○○○●●●●

● Current color
○ Back ground color

【Related function】

glLineWidth function, **glBackColor** function, **glEnable** function

【Acquire the parameter】

It can acquire the current broken line pattern using **glGet*** function.

The argument is **GL_LINE_STIPPLE_PATTERN**.

6.9. Texture

6.9.1. glTexImage2D Configure the 2D texture image

【Interface】

GLint **glTexImage2D**(GL_CTX **ctx*, GLenum *target*, GLint *memory*, GLint *level*,
GLsizei *width*, GLsizei *height*, GLint *border*,
GLenum *format*, GLenum *type*, GLuint *texadrs*
const GLvoid **pixels*)

【Argument】

ctx	Pointer to 3DGL context
target	It specifies the target・texture.（can only specify **GL_TEXTURE_2D**）
memory	Type of memory, which loads the texture image
level	Texture・MIP map level（can only specify 0）
width	Width of the texture image（total pixels）
height	Height of the texture image（total lines）
border	Width of the bound（can only specify 0）
format	Format of the pixel data（unused）
type	Format of pixel data of texture image （can only specify **GL_USHORT_1_5_5_5**）
texadrs	Storage location address of texture image
pixels	Pointer to texture image to load

【Returned value】

GL_TRUE	Normal end
GL_FALSE	Abnormal end

【Description】

It loads the texture image to the internal texture memory of the graphics controller or the graphics memory.

It can load multi texture image due to specifying the address.

If it refers the texture image or switches it, it specifies the reference texture by **glTexMemoryMode** function.

Specify the type of the memory, which loads the texture image, for *memory*. It shows the type of the memory, which can be specified in Table6.9.1.

Specify the width and the height of each texture image for *width, height*. There are constraints on the size of the texture image by the graphics controller. Refer to *Appendix A The comparative chart on the graphics controller*.

format is not used, but specify **GL_RGBA**.

Specify the loading address of the memory for *texadrs.* If the loading location is the internal texture memory, specify the offset address from top of the internal texture memory. If the loading location is the graphics memory, specify the offset address from top of the graphics memory. If it specifies **GL_TEXTURE_INT_DYNAMIC** for *memory*, specify the offset address of the graphics memory for the loading memory address.

Specify the area, which stores the texture image for *pixels*.

In order to enable the texture process, it specifies **GL_TEXTURE_2D** by **glEnable** function.

In order to disable the texture process, it specifies **GL_TEXTURE_2D** by **glDisable** function.

【Notice】

If it uses **GL_TEXTURE_INT_DYNAMIC** in case of that, it loads the texture image into the internal texture image specifying **GL_TEXTURE_INT** in advance, it might break down the texture data, which is loaded in advance.

【Supplement】

target, *level*, *border*, *format*, *type* are arguments, which are prepared for the extension. Use only defined configured value currently. If it specifies another configured value, the motion is an inconstancy.

【Related function】

glTexMemoryMode function, **glAlpha** function, **glTexEnvi** function, **glTexParameter*** function

Table6.9.1 Symbolic constant showing type of texture buffer

Symbolic constant	Meaning
GL_TEXTURE_INT	It uses the internal texture memory. There is the graphics controller, which does not have the internal texture memory, depend on the type. Refer to Appendix A The comparative chart of the graphics controller.
GL_TEXTURE_EXT	It uses the graphics memory. In case of loading multi texture images, it can switch the texture to refer by **glTexMemoryMode** function.
GL_TEXTURE_INT_DYNAMIC	It performs the dynamic loading to the internal texture memory. It transfers the texture image, which is loaded to the graphics memory, and then performs the drawing process. Therefore, specify the address of the graphics memory for *teadrs* of **glTexImage2D** function.

6.9.2. glTexParameter* Configure the texture・parameter

【Interface】

void **glTexParameteri**(GL_CTX **ctx*, GLenum *target*,
GLenum *pname*, const GLint *param*)
void **glTexParameterfv**(GL_CTX **ctx*, GLenum *target*,
GLenum *pname*, const GLfloat **params*)
void **glTexParameteriv**(GL_CTX **ctx*, GLenum *target*,
GLenum *pname*, const GLint **params*)

【Argument】

ctx	Pointer to 3DGL context
target	Specify **GL_TEXTURE_2D**
pname	Type of the texture・parameter
param	Configured value of the texture・parameter
params	Pointer to the area, which stores the configured value of texture・parameter

【Returned value】

None

【Description】

It configures the parameter value of the texture mapping.

It specifies one of the parameter type shown in Table6.9.2 (next page) for *pname*, then it specifies the configured value for *param* (**glTexParameteri** function). Also, specify the area, which stores the configured value for *params* (**glTexParameterfv** function and **glTexParameteriv** function).

【Related function】

glTexEnvi function, **glTexImage2D** function

【Acquire the parameter】

It can acquire the texture・parameter using **glGetTexParameter*** function.

Table6.9.2 Symbolic constant showing texture・parameter

Symbolic constant	Meaning
GL_TEXTURE_FILTER	It specifies the texture filter mode. It can specify **GL_NEAREST** or **GL_LINEAR** for the configured value.
GL_TEXTURE_WRAP_S **GL_TEXTURE_WRAP_T**	It specifies the wrap method of the texture coordinate(s, t). It can specify one of **GL_CLAMP** 、 **GL_REPEAT** 、 **GL_BORDER** for the configured value.
GL_TEXTURE_BORDER_COLOR	It configures the border color in case of specifying **GL_BORDER** for the texture wrapping parameter. The border color must be configured, before it specifies the wrap method of the texture coordinate(s, t). It stores the border color in order of R, G, B, A for the area specifying by *param.* The configuration of **GL_TEXTURE_BORDER_COLOR** is not performed by **glTexParameteri** function.

6.9.3. glTexEnvi Configure the texture environmental parameter

【Interface】

void **glTexEnvi**(GL_CTX **ctx*, GLenum *target*, GLenum *pname*, GLint *param*)

【Argument】

ctx	Pointer to 3DGL context
target	Texture environment（can specify only **GL_TEXTURE_ENV**）
pname	Type of the texture environmental parameter
param	Configured value of the texture environmental parameter

【Returned value】

None

【Description】

It configures the texture environmental parameter. In the texture environmental parameter, it configures the texture blending mode and the texture alpha blending. (Refer to *2.8.6 Texture blending mode* and *2.8.7 Texture alpha blending*.)

It specifies the type of the texture environmental parameter to configure for *pname*, it specifies the configured value of the texture environmental parameter, which is specified by *pname*, for *param*.

It shows the type of the texture environmental parameter and the configured value, which is able to be configured for *pname* and *param*, in Table6.9.3 (next page).

The initial configuration of **GL_TEXTURE_ALPHA_MODE** is **GL_MODULATE**.

In **GL_TEXTURE_ALPHA_MODE**, the configured value is enabled, only when the alpha blending is enabled. There is no initial configuration.

【Related function】

glTexImage2D function, **glTexParameter*** function, **glEnable** function

【Acquire the parameter】

It can acquire the texture environmental parameter using **glGetTexEnv*** function.

Table6.9.3 Symbolic constant showing the texture environmental parameter

Symbolic constant specifying in *pname*	Configured value of *param*	Meaning
GL_TEXTURE_ENV_MODE	**GL_DECAL**	Execute DECAL
	GL_MODULATE	Execute MODULATE
	GL_STENCIL	Execute STENCIL
GL_TEXTURE_ALPHA_MODE	**GL_ALPHA_ALL**	Execute ALL
	GL_ALPHA_STENCIL	Execute STENCIL
	GL_ALPHA_STENCIL_ALPHA	Execute STENCILALPHA

6.9.4. glTexCoord2* Configure the texture coordinate

【Interface】

void **glTexCoord2f** (GL_CTX **ctx*, GLfloat *s*, GLfloat *t*)

void **glTexCoord2fv**(GL_CTX **ctx*, const GLfloat **tex*)

【Argument】

ctx	Pointer to 3DGL context
s	Texture s-coordinate
t	Texture t-coordinate
tex	Pointer to the area, which stores the texture coordinate(s, t)

【Returned value】

None

【Description】

It configures the current texture coordinate.

Store the configured value in order of s, t in the area specified by *tex*.

This function could be specified between **glBegin** function and **glEnd** function.

It is normally used in combination with specifying the apex coordinate.

【Related function】

glVertex3* function

【Acquire the parameter】

It can acquire the current texture coordinate using **glGet*** function. The argument is **GL_CURRENT_TEXTURE_COORDS**.

6.9.5. glTexMemoryMode Specify the texture image

【Interface】

void **glTexMemoryMode**(GL_CTX **ctx*,
GLenum *memory*, GLsizei *width*, GLsizei *height*, GLuint *texadrs*)

【Argument】

ctx	Pointer to 3DGL context
memory	Memory type, which stores the texture image to use
width	Wide of the texture image to use (total pixels)
height	Height of the texture image to use (total lines)
texadrs	Address, which loads the texture image to use

【Returned value】

None

【Description】

It specifies the texture image, which is used for the texture mapping.

It refers the texture image, which is loaded into the texture buffer showing *texadrs*, in the following texture mapping. It is necessary to load the texture image into the graphics memory or the internal graphics memory by **glTexImage2D** function in advance.

Specify the memory type shown in Table6.9.1 (shown in *6.9.1 glTexImage2D*) for *memory*. If it specifies **GL_TEXTURE_INT_DYNAMIC**, it transfers the internal texture memory, which is loaded into the graphics memory by **glTexImage2D** in advance, it sets the referencing memory up as the internal texture memory.

It specifies the width and the height of the texture image for *width, height*.

It specifies the offset address from each top area for *texadrs*, it is depending on whether the referring texture image is the internal texture memory or the graphics memory.

【Related function】

glTexImage2D function

6.10. Buffer control

6.10.1. glClearBuffer Clear the buffer

【Interface】

void **glClearBuffer**(GL_CTX **ctx*, GLbitfield *mask*)

【Argument】

ctx	Pointer to 3DGL context
mask	Buffer to erase

【Returned value】

None

【Description】

It clears the polygon drawing buffer and the depth buffer.

Specify the symbolic constant for *mask* shown in below table. Both of them could be specified by the logical addition (OR).

Table6.10.1 Symbolic constant showing the clearing target buffer

Symbolic constant	Meaning
GL_POLYGON_BUFFER_BIT	It clears the polygon drawing buffer.
GL_DEPTH_BUFFER_BIT	It clears the depth buffer (Z buffer).

【Complement】

Currently, it does not have the drawing function, which uses the polygon drawing buffer.

【Related function】

glCreateBuffer function

6.10.2. glCreateBuffer Create the buffer

【Interface】

void **glCreateBuffer**(GL_CTX *ctx*, GLbitfield *mask*, GLuint *adrs*)

【Argument】

ctx	Pointer to 3DGL context
mask	Buffer to create
adrs	Origin address of the buffer to create

【Returned value】

None

【Description】

It creates the polygon drawing buffer and the depth buffer.

Specify the symbolic constant for *mask* shown in below table. Both of them could be specified by the logical addition (OR).

Table6.10.2 Symbolic constant showing the creating target buffer

Symbolic constant	Meaning
GL_POLYGON_BUFFER_BIT	It creates the polygon drawing buffer.
GL_DEPTH_BUFFER_BIT	It creates the depth buffer (Z buffer). It is required 2 byte per 1 pixel.

【Complement】

Currently, it does not have the drawing function, which uses the polygon drawing buffer.

【Related function】

glClearBuffer function

6.10.3. glDepthMask Writing control to the depth buffer

【Interface】

void **glDepthMask**(GL_CTX **ctx*, GLboolean *flag*)

【Argument】

ctx	Pointer to 3DGL context
flag	Writing enable/ disable to the depth buffer

【Returned value】

None

【Description】

It specifies whether the depth buffer is enabled to be written or not.

If *flag* is 0, writing to the depth buffer is invalid. Except this, it is valid.

The initial configuration of the writing to the depth buffer is enabled.

【Related function】

glDepthFunc function, **glDepthRange** function

6.10.4. glDepthFunc Configure the depth test method

【Interface】

void **glDepthFunc**(GL_CTX **ctx*, GLenum *func*)

【Argument】

ctx	Pointer to 3DGL context
func	Depth test method

【Returned value】

None

【Description】

It configures the comparative method of z value on the depth test.

Specify one of Table6.10.4 for *func*. The initial value is **GL_LESS**.

The depth test is invalid in the initial configuration.

In order to enable the depth test, it specifies **GL_DEPTH_TEST** by **glEnable** function.

In order to disable the depth test, it specifies **GL_DEPTH_TEST** by **glDisable** function.

Table6.10.4 Symbolic constant showing the depth test method

Symbolic constant	Meaning
GL_NEVER	It does not always draw.
GL_LESS	It draws, if z value is less than saved z value. (initial value)
GL_EQUAL	It draws, if z value is equal to saved z value.
GL_LEQUAL	It draws, if z value is less than saved z value or equal to it.
GL_GREATER	It draws, if z value is larger than saved z value.
GL_NOTEQUAL	It draws, if z value is not equal to saved z value.
GL_GEQUAL	It draws, if z value is larger than saved z value or equal to it.
GL_ALWAYS	It always draws.

【Related function】

glDepthRange function, **glEnable** function, **glDisable** function

【Acquire the parameter】

It can acquire the comparative method of z value on the current depth test using **glGet*** function. The argument is **GL_DEPTH_FUNC**.

6.10.5. glClear Clear the view port area

【**Interface**】

void **glClear**(GL_CTX **ctx*)

【**Argument**】

ctx Pointer to 3DGL context

【**Returned value**】

None

【**Description**】

It clears the view port area by the color configured by **glClearColor** function.

The value configured by **glViewport** function is used for the view port area. For that reason, it is necessary to perform **glClearColor** function and **glViewport** function in advance.

【**Related function**】

glClearColor function, **glViewport** function

6.11. View port

6.11.1. glDepthRange Configure the clipping on the device coordinate transformation

【Interface】

void **glDepthRange**(GL_CTX **ctx*, GLclampd *near*, GLclampd *far*)

【Argument】

ctx	Pointer to 3DGL context
near	z-coordinate on the front clip
far	z-coordinate on the back clip

【Returned value】

None

【Description】

It configures the front clip face and the back clip face on the normalization device coordinate transformation.

Set the z-coordinate up as -1.0~1.0 for *near* and *far*.

The initial value of *near* is 0, the initial value of *far* is 1.

【Related function】

glDepthFunc function, **glViewport** function

【Acquire the parameter】

It can acquire the current depth value using **glGet*** function. The argument is **GL_DEPTH_RANGE**.

6.11.2. glViewport Configure the view port

【Interface】

void **glViewport**(GL_CTX **ctx*, GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*)

【Argument】

ctx	Pointer to 3DGL context
x	x-coordinate direction offset on the device coordinate transformation
y	y-coordinate direction offset on the device coordinate transformation
width	Width of the view port（total pixels）
height	Height of the view port（total lines）

【Returned value】

None

【Description】

It configures the parameter, which converts the normalization device coordinate to the device coordinate.

The device coordinate (Xd, Yd) is calculated in the following in response to the normalization device coordinate (Xnd, Ynd).

$$Xd = (Xnd + 1) \times (width / 2) + x$$

$$Yd = (Ynd + 1) \times (height / 2) + y$$

The initial value of *x, y* is (0, 0), and then the origin of the normalization device coordinate corresponds to the center of the view port.

【Related function】

glDepthRange function

【Acquire the parameter】

It can acquire the current view port using **glGet*** function. The argument is **GL_VIEWPORT**.

6.12. Enable／Disable of the function

6.12.1.glEnable, glDisable Enable／Disable of the function

【Interface】

void **glEnable**(GL_CTX **ctx*, GLenum *cap*)

void **glDisable**(GL_CTX **ctx*, GLenum *cap*)

【Argument】

ctx	Pointer to 3DGL context
cap	Symbolic constant showing the specific function

【Returned value】

None

【Description】

glEnable function enables the function specified by *cap.*

glDisable function disables the function specified by *cap.*

It specifies one of the symbolic constant showing in Table6.12.1 (next page) for *cap*.

【Related function】

glAlpha* function, **glNormal3*** function, **glLight*** function, **glLineWidth** function, **glLineStipple** function

glTexImage2D function, **glDepthFunc** function, **glCullFace** function

Table6.12.1 Symbolic constant, which shows the function to enable/ disable

Symbolic constant	Meaning
GL_BLEND	Alpha blending
GL_CULL_FACE	Shading
GL_DEPTH_TEST	Depth test
GL_LIGHT*n* （***n*** **：** 0~7）	Lighting process of n light
GL_LIGHTING	Lighting process
GL_LINE_ENDPOINT	End point drawing of the line
GL_LINE_SMOOTH	Anti-aliasing
GL_LINE_STIPPLE	Broken line drawing
GL_LINE_STIPPLE_OFFSET	Offset clear of the broken line pattern (It draws the broken line from the beginning.)
GL_NORMALIZE	Normalization of the normal vector
GL_PERSPECTIVE	Texture correction
GL_TEXTURE_2D	Texture mapping

6.13. Shading

6.13.1. glFrontFace Configure the front face

【Interface】

void **glFrontFace**(GL_CTX **ctx*, GLenum *mode*)

【Argument】

ctx	Pointer to 3DGL context
mode	Front direction

【Returned value】

None

【Description】

It configures the front face for the shading.

The front face is a triangle face looking from the eye view. When the shading is enabled, it draws only the front face and it does not draw the back face. (It is possible to draw only the back face by **glCullFace** function.)

The configuration of the front face is determined by the assignment order of the apex of a triangle. Specify one of Table6.13.1 for *mode*. The initial value is **GL_CCW**.

In order to enable the shading, it specifies **GL_CULL_FACE** by **glEnable** function.

In order to disable the shading, it specifies **GL_CULL_FACE** by **glDisable** function.

Table6.13.1 Symbolic constant specifying the front face

Symbolic constant	Meaning
GL_CW	Clockwise
GL_CCW	Counter clockwise（Initial value）

【Related function】

glCullFace function, **glLightModel*** function

【Acquire the parameter】

It can acquire the current front face direction by **glGet*** function. The argument is **GL_FRONT_FACE**.

6.13.2. glCullFace Configure the shading face

【Interface】

void **glCullFace**(GL_CTX *ctx*, GLenum *mode*)

【Argument】

ctx	Pointer to 3DGL context
mode	Shading face

【Returned value】

None

【Description】

In the shading, it specifies the target of the shading, the front direction or the back direction.

Specify one of Table6.13.2 for *mode.* The initial value is **GL_BACK**.

In order to enable the shading, it specifies **GL_CULL_FACE** by **glEnable** function.

In order to disable the shading, it specifies **GL_CULL_FACE** by **glDisable** function.

Table6.13.2 Symbolic constant specifying the target face of the shading

Symbolic constant	Meaning
GL_FRONT	Front direction
GL_BACK	Back direction（Initial value）

【Related function】

glFrontFace function, **glEnable** function, **glDisable** function

【Acquire the parameter】

It can acquire the direction of the shading using **glGet*** function.

The argument is **GL_CULL_FACE_MODE**.

6.14. Acquire the parameter

6.14.1. glGet* Acquire the parameter

【Interface】

void **glGetBooleanv**(GL_CTX **ctx*, GLenum *pname*, GLboolean **params*)

void **glGetDoublev**(GL_CTX **ctx*, GLenum *pname*, GLdouble **params*)

void **glGetFloatv**(GL_CTX **ctx*, GLenum *pname*, GLfloat **params*)

void **glGetIntegerv**(GL_CTX **ctx*, GLenum *pname*, GLint **params*)

【Argument】

ctx	Pointer to 3DGL context
pname	Type of the parameter to acquire
params	Pointer to the area, stores the parameter value acquired

【Returned value】

None

【Description】

It acquires the various parameter values.

Specify the parameter value to acquire for *pname*. The addressable parameter is shown in Table6.14.1 (recording in end of 6.14.1).

Specify the area, which stores the value acquired by *params.* The parameter specified by *pname* has multi element values (such as color of red, green, blue), each element value is stored in order of Table6.14.1.

In each **glGet*** function, the data types of the acquired parameter value are various. Each function is explained in the following.

- **glGetBooleanv**

  It returns the acquired parameter value by **GL_TRUE** or **GL_FALSE**.

  If the parameter specified by *pname* is the integer type or the floating point type, it returns **GL_FALSE** when the value is 0, it returns **GL_TRUE** in others value.

- **glGetDoublev**

  It returns the acquired value by the GLdouble type.

  If the parameter specified by *pname* is the integer type or the logical value (**GL_TRUE** or

GL_FALSE), it returns the value, which is transformed in the GLdouble type.

- **glGetFloatv**

It returns the acquired parameter value by the GLfloat type.

If the parameter specified by *pname* is the integer type or the logical value (**GL_TRUE** or **GL_FALSE**), it returns the value, which is transformed in the GLfloat type.

- **glGetIntegerv**

It returns the acquired parameter value by the GLing type.

The following parameter of the floating point type is transformed in the value, which the linear mapping is performed, [-1.0, 1.0] into [-2^{31}-1,2^{31}-1]. The others parameter of the floating point type is transformed to the integer type.

・Current color（**GL_CURRENT_COLOR**）

・View port area clear color（**GL_COLOR_CLEAR_VALUE**）

・Current blend index（**GL_CURRENT_ALPHA**）

・Normal vector（**GL_CURRENT_NORMAL**）

・Depth range（**GL_DEPTH_RANGE**）

The parameter of the logical value (**GL_TRUE** or **GL_FALSE**) is returned in the value itself.

【Related function】

glGetError function, **glGetLight*** function, **glGetMatrial*** function, **glGetTexEnv*** function, **glGetTexParameter*** function

Table6.14.1 Symbolic constant showing parameter type to acquire the information

Symbolic constant	Meaning
GL_ATTRIB_STACK_DEPTH	Total of current property stack
GL_BLEND	Logical value showing enable/disable of alpha blending
GL_COLOR_CLEAR_VALUE	Color value, which is used to paint the rectangle (red, green, blue of three value) (*1)
GL_CULL_FACE	Logical value showing enable/disable of the shading
GL_CULL_FACE_MODE	Symbolic constant showing the triangle face of the shading
GL_CURRENT_ALPHA	Current alpha blending index
GL_CURRENT_COLOR	Current color value (red, green, blue of three value) (*1)
GL_CURRENT_NORMAL	Current normal vector value (x,y,z of three value)
GL_CURRENT_TEXTURE_COORDS	Current texture coordinate (s, t of two value)
GL_DEPTH_FUNC	Symbolic constant showing the depth test function
GL_DEPTH_RANGE	Front clip face and rear clip face (near, far of two value)
GL_DEPTH_TEST	Logical value showing enable/disable of depth test
GL_DEPTH_WRITEMASK	Logical value showing enable/disable of depth buffer write
GL_FRONT_FACE	Symbolic constant showing clockwise/counter-clockwise of front direction of a triangle
GL_LIGHT*n* （***n***：0~7）	Logical value showing enable/disable of the light *n*
GL_LIGHTING	Logical value showing enable/disable of the lighting
GL_LIGHT_MODEL_AMBIENT	Brightness of ambient light of the full view (red, green, blue, alpha of four value)

（continued）

*1： The color value is transformed internally into range of [0,255] and kept. Therefore, the acquired value might be different from the value specified by **glColor3f** function.

Table6.14.1 Symbolic constant showing parameter type to acquire the information（continued）

Symbolic constant	Meaning
GL_LIGHT_MODEL_LOCAL_VIEWER	Logical value showing enable/disable of calculation of mirror reflection direction
GL_LIGHT_MODEL_TWO_SIDE	Logical value showing if the lighting calculation of front direction and rear direction of a triangle is performed severally.
GL_LINE_ENDPOINT	Logical value showing enable/disable of the end point drawing of the line
GL_LINE_SMOOTH	Logical value showing enable/disable of anti aliasing process of the line
GL_LINE_STIPPLE	Logical value showing enable/disable of the broken line drawing
GL_LINE_STIPPLE_OFFSET	Logical value showing enable/disable of the offset process of the broken line pattern
GL_LINE_STIPPLE_PATTERN	Value of the broken line pattern
GL_LINE_WIDTH	Line width
GL_LINE_WIDTH_GRANULARITY	Differential of the line width, which is adjacent anti-aliasing line.
GL_LINE_WIDTH_RANGE	Maximum width and minimum width of anti-aliasing line
GL_MATRIX_MODE	Symbolic constant showing the matrix stack of the current stack on the matrix process
GL_MAX_ATTRIB_STACK_DEPTH	Total of maximum property stack
GL_MAX_LIGHTS	Total of maximum light
GL_MAX_MODELVIEW_STACK_DEPTH	Total of maximum stack of the model view matrix
GL_MAX_PROJECTION_STACK_DEPTH	Total of maximum stack of the projection matrix
GL_MAX_TEXTURE_SIZE	Maximum width or maximum height, which is enable to use for the texture image
GL_MAX_VIEWPORT_DIMS	Maximum width and maximum height, which is enable to specify the view port
GL_MODELVIEW_MATRIX	Beginning 16 value of the model view matrix stack (4x4 matrix)
GL_MODELVIEW_STACK_DEPTH	Total of the model view matrix stack

（continued）

Table6.14.1 Symbolic constant showing parameter type to acquire the information（continued）

Symbolic constant	Meaning
GL_NORMALIZE	Logical value showing enable/disable of normalization process of the normal vector
GL_PERSPECTIVE	Logical value showing enable/disable of the texture correction
GL_PROJECTION_MATRIX	Beginning 16 value of the projection matrix stack (4x4 matrix)
GL_PROJECTION_STACK_DEPTH	Total of the projection matrix stack
GL_RENDER_MODE	Symbolic constant showing the rendering mode (**GL_REDER** is fixed)
GL_SHADE_MODEL	Symbolic constant showing the shading mode
GL_TEXTURE_2D	Logical value showing enable/disable of 2D texture process
GL_TEXTURE_ENV_MODE	Symbolic constant showing the texture blending mode selected currently
GL_TEXTURE_ALPHA_MODE	Symbolic constant showing the texture alpha blending mode selected currently
GL_VIEWPORT	The device coordinate of the view port (x,y), width, height of four value

6.14.2. glGetError Acquire the error information

【Interface】

GLenum **glGetError**(GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context

【Returned value】

Symbolic constant of the error information

【Description】

It acquires the error information.

This function returns one of the below table.

Table6.14.2 Symbolic constant showing the error information

Symbolic constant	Meaning
GL_NO_ERROR	No error
GL_INVALID_ENUM	Invalid value is specified for the argument
GL_INVALID_VALUE	Outside value is specified for the argument
GL_INVALID_OPERATION	The operation is impossible
GL_STACK_OVERFLOW	It occurs the stack・over flow
GL_STACK_UNDERFLOW	It occurs the stack・under flow

6.14.3. glGetLight* Acquire the lighting parameter value

【Interface】

void **glGetLightfv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, GLfloat **params*)

void **glGetLightiv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, GLint **params*)

void **glGetLightubv**(GL_CTX **ctx*, GLenum *light*, GLenum *pname*, GLubyte **params*)

【Argument】

ctx	Pointer to 3DGL context
light	Light
pname	Lighting parameter
params	Pointer to area, stores the lighting parameter value

【Returned value】

None

【Description】

It acquires the specified lighting parameter value.

Specify one of **GL_LIGHT0～GL_LIGHT7** for *light*.

The acquired value is configured in *params*. The value is singular number or plural number depended on the lighting parameter type.

The addressable lighting parameter is shown in Table6.14.3 (next page).

【Related function】

glLight* function

Table6.14.3 Symbolic constant showing the lighting parameter

Symbolic constant	Meaning	
GL_AMBIENT	(R, G, B, A), four value showing the brightness of the ambient light of the light	*1
GL_DIFFUSE	(R, G, B, A), four value showing the brightness of the diffused reflection of the light	*1
GL_SPECULAR	(R, G, B, A), four value showing the brightness of the mirror reflection of the light	*1
GL_POSITION	(x,y,z,w), four value showing the position of the light	
GL_SPOT_DIRECTION	(x,y,x), three value showing the direction of the light	
GL_SPOT_EXPONENT	Brightness distribution index of the light	
GL_SPOT_CUTOFF	Cutoff of the light	
GL_CONSTANT_ATTENUATION	Fixed attenuation constant of the light	
GL_LINEAR_ATTENUATION	Linear attenuation constant of the light	
GL_QUADRATIC_ATTENUATION	Quadratic attenuation constant of the light	

*1 If it acquires the value by the integer type, the linear mapping is performed, which is corresponding to the maximum value of the integer type is 1.0.

6.14.4. glGetMaterial* Acquire the material parameter value

【**Interface**】

void **glGetMaterialfv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, GLfloat **params*)

void **glGetMaterialiv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, GLint **params*)

void **glGetMaterialubv**(GL_CTX **ctx*, GLenum *face*, GLenum *pname*, GLubyte **params*)

【**Argument**】

ctx	Pointer to 3DGL context
face	Specified front direction/ back direction of a triangle
pname	Material parameter
params	Pointer to the area, which stores the acquired material parameter value

【**Returned value**】

None

【**Description**】

It acquires the material parameter value.

Either **GL_FRONT** (front direction) or **GL_BACK** (back direction) specifies if it acquires the material parameter of the front direction or back direction of a triangle for *face.*

Specify the type of the material parameter for *pname*. It shows the addressable material parameter in Table6.14.4.

The acquired value is stored in *params.*

Table6.14.4 **Symbolic constant showing the material parameter**

Symbolic constant	Meaning	
GL_AMBIENT	(R,G,B,A), four value showing reflection rate of the ambient light of the material	*1
GL_DIFFUSE	(R,G,B,A), four value showing reflection rate of diffused light of the material	*1
GL_SPECULAR	(R,G,B,A), four value showing reflection rate of mirror light of the material	*1
GL_EMISSION	(R,G,B,A), four value showing the radiance of the material	*1
GL_SHININESS	Mirror brightness distribution index of the material	

*1 If it acquires the value by the integer type, the linear mapping is performed, which is corresponding to the maximum value of the integer type is 1.0.

【Related function】

glMaterial* function

6.14.5. glGetTexEnviv Acquire the texture environmental parameter value

【Interface】

void **glGetTexEnviv**(GL_CTX **ctx*, GLenum *target*, GLenum *pname*, GLint **params*)

【Argument】

ctx	Pointer to 3DGL context
target	Specify **GL_TEXTURE_ENV**.
pname	Texture environmental parameter
params	Pointer to the area, stores the texture environmental parameter value

【Returned value】

None

【Description】

It acquires the texture environmental parameter value.

It specifies the type of the texture environmental parameter for *pname*.

It specifies the area, stores the acquired texture environmental parameter value for *params*.

The addressable texture environmental parameter is the following below table.

Table6.14.5 Symbolic constant showing the texture environmental parameter

Symbolic constant	Meaning
GL_TEXTURE_ALPHA_MODE	Texture alpha blending mode
GL_TEXTURE_ENV_MODE	Texture blending mode
GL_TEXTURE_MEMORY_MODE	Texture buffer type

【Related function】

glTexEnv* function

6.14.6. glGetTexParameter Acquire the texture・parameter value

【Interface】

void **glGetTexParameterfv**(GL_CTX **ctx*, GLenum *target*, GLenum *pname*,
GLfloat **params*)

void **glGetTexParameteriv**(GL_CTX **ctx*, GLenum *target*, GLenum *pname*,
GLint **params*)

【Argument】

ctx	Pointer to 3DGL context
target	Target・texture（can specify only **GL_TEXTURE_2D**）
pname	Type of the texture・parameter
params	Pointer to the area, stores the texture・parameter value

【Returned value】

None

【Description】

It acquires the texture・parameter value.

Specify the type of the texture・parameter for *pname*. The addressable texture・parameter is the following Table6.14.6.

Specify the area, which stores the acquired texture・parameter for *params*.

Table6.14.6 Texture parameter showing the target texture・parameter

Symbolic constant	Meaning
GL_TEXTURE_FILTER	Texture filter format
GL_TEXTURE_WAP_S	Texture wrap method in s-coordinate direction
GL_TEXTURE_WAP_T	Texture wrap method in t-coordinate direction
GL_TEXTURE_BORDER_COLOR	Border color value

【Related function】

glTexParameter* function

6.15. Push and pop of the various property

6.15.1. glPushAttrib, glPopAtrib Save and restore the property

【Interface】

void **glPushAttrib**(GL_CTX **ctx*, GLbitfield *mask*)

void **glPopAttrib**(GL_CTX **ctx*)

【Argument】

ctx	Pointer to 3DGL context
mask	Target property

【Returned value】

None

【Description】

glPushAttrib function saves the various properties on the property stack.

glPopAttrib function recovers the property, which is saved on the property stack.

Specify the property to save for **mask**. The specification is performed by the logical addition of the symbolic constant, which shows the property to save.

The property stack is an empty in the initial condition.

If it saves the property with the property stack being full or it tries to recovery with the property stack being empty, it occurs an error.

【Acquire the parameter】

It can acquire the current total property stacks and the maximum total stacks by **glGet*** function.

The argument is **GL_ATTRIB_STACK_DEPTH** and **GL_MAX_ATTRIB_STACK_DEPTH**.

Table6.15.1 Symbolic constant showing the property for the targeting save・recovery

Symbolic constant	Target property
GL_CURRENT_BIT	Current color value Current blend index Current normal vector Current texture coordinate
GL_LINE_BIT	Enable/disable of anti-aliasing of line Enable/disable of broken line drawing Enable/disable of end point of line Broken line pattern Line width
GL_POLYGON_BIT	Enable/disable of shading Face of shading (front/back direction) Definition of front face (clockwise/counter clockwise)
GL_LIGHTING_BIT	Enable/disable of lighting Enable/disable of each lighting Brightness of environment light, diffused light, mirror light Direction, position, brightness distribution index, cutoff of each light Fixed, linear, quadric attenuation constant of each light Reflection rate of environment light, diffused light, mirror light Radiation color Mirror brightness distribution index Enable/disable of back direction material Calculation method of mirror reflection angle Shading mode
GL_DEPTH_BUFFER_BIT	Enable/disable of depth test Depth test method Enable/disable of depth buffer write
GL_VIEWPORT_BIT	Depth range Origin of the view port and the range

（continued）

Table6.15.1 Symbolic constant showing the property for the targeting save・recovery (continued)

Symbolic constant	Target property
GL_TRANSFORM_BIT	Matrix stack of the current target Enable/disable of normalization process of normal vector
GL_ENABLE_BIT	Enable/disable of alpha blending Enable/disable of shading Enable/disable of depth test Enable/disable of lighting Enable/disable of each light Enable/disable of anti-aliasing of line Enable/disable of broken line drawing Enable/disable of end point of line Enable/disable of normalization process of normal vector Enable/disable of texture mapping Enable/disable of texture correction
GL_TEXTURE_BIT	Enable/disable of texture mapping Size of texture image (width, height) Texture image reference address Enable/disable of texture correction Border color Texture filter mode Texture wrap mode Texture environmental parameter
GL_ALL_ATTRIB_BITS	Special mask for preserving all property value

Appendix A　The comparative chart of the graphics controller

TableA The comparative chart of the graphics controller

		Graphics controller		
		MB86291(*1)	MB86292(*2)	MB86293 after(*3)
Maximum drawing frame size(*4)		4096×4096	4096×4096	4096×4096
Capacity of the graphics memory		2MB(internal)	Max 32MB	Max 32MB
Enable texture image size (*5)（width, height）	Using internal texture memory	4/8/16/32/64	4/8/16/32/64	
	Using graphics memory	4/8/16/32/64 /128/256	4/8/16/32/64 /128/256	4/8/16/32/64 /128/256/512/ /1024/2048/4096
Texture coordinate range		-512～511	-512～511	-8192～8191

*1）MB86921A、MB86291S included

*2）MB86292S included

*3）MB86293 and MB86294。

*4）The settable drawing frame size is within the capacity of the graphics memory.

*5）It can use the different size of the width and the height.